École doctorale SPI Lille Nord-De-France

THÈSE

pour obtenir le grade de docteur délivré par

l'Université du Littoral Côte d'Opale

Spécialité doctorale "Informatique"

présentée et soutenue publiquement par

Christopher Herbez

le 21 novembre 2016

Parallélisation massive de dynamiques spatiales : contribution à la gestion durable du mildiou de la pomme de terre

Directeur de thèse : **Éric Ramat**

Co-encadrant de thèse : **Gauthier Quesnel**

Jury

Alexis Drogoul,	Rapporteur	Senior Researcher HdR (UMI UMMISCO 209)
Jean-François Santucci,	Rapporteur	Professeur (SPE, Université de Corse)
Claudia Frydman,	Examinateur	Professeur (LSIS, Université de Marseille)
Vincent Chevrier,	Examinateur	Professeur (Loria, Université de Lorraine)
Jean Noël Aubertot	Examinateur	Directeur de recherche - HdR (INRA, Toulouse)
Éric Ramat,	Directeur de thèse	Professeur (LISIC, ULCO)
Gauthier Quesnel,	Co-encadrant	Chargé de recherche (INRA, Toulouse)

Laboratoire d'Informatique, Signal et Image de la Côte d'Opale – EA 4491

Maison de la Recherche Blaise Pascal – Centre Universitaire de la Mi-Voix

50 rue Ferdinand Buisson – B.P. 719, 62228 Calais Cedex, France

Remerciements

Je remercie chaleureusement toutes les personnes qui m'ont aidé pendant l'élaboration de cette thèse et principalement mon directeur Monsieur Éric Ramat, pour son intérêt, ses conseils et son soutien tout au long de cette thèse. Il a su trouver le juste équilibre entre encadrement et autonomie me laissant ainsi l'opportunité d'exprimer tout mon potentiel. Je le remercie également pour sa grande disponibilité et ses nombreux conseils, notamment lors de la rédaction de cette thèse. Je me sens privilégié d'avoir eu l'opportunité d'être dirigé par un chercheur de son rang.

Je tiens également à remercier l'intégralité des membres du LISIC, mon laboratoire d'accueil. Tout particulièrement Émilie Poisson Caillault, mon ancienne directrice des études de master, qui a su me donner goût à la recherche. Je remercie par la même occasion l'Université du Littoral Côte d'Opale pour ces huit années d'études supérieures de qualité.

Cette thèse fut réalisée en étroite collaboration avec les membres du milieu agronomique et agricole. Je tiens par cette présente à remercier tout particulièrement monsieur Michel Denis, pour les moyens mis en place pour que cette thèse puisse voir le jour. Je tiens également à remercier monsieur Alexandre Rivenet, ingénieur agricole, pour nous avoir fourni les données liées à ses cultures et pour son retour d'expérience. Je remercie dans leur ensemble les membres de l'INRA de Toulouse et de Ploudaniel, mais aussi les membres de la plate-forme RECORD et ceux de l'ACVNPT. Ce fut un réel plaisir de pouvoir côtoyer toutes ces personnes et de pouvoir profiter de leur savoir.

Ces travaux de thèse m'ont permis d'entrer dans la communauté DEVS francophone, regroupée sous l'appellation RED. Je tiens à remercier chacun de ces membres, notamment pour leur accueil lors des journées DEVS francophone de 2016 (JDF16). Au cours de cette conférence, j'ai eu l'immense honneur de rencontrer B. P. Zeigler, créateur du formalisme DEVS et de la plupart de ces extensions. C'est un véritable plaisir de pouvoir collaborer avec des chercheurs de cette qualité.

Sur un plan personnel, je tiens à remercier ma famille et tout particulièrement ma femme pour sa patience, ses encouragements et pour être une source d'inspiration au quotidien. Pour finir, je tiens à dédier cette thèse à mes parents, simples ouvriers aux cœurs d'ors, qui m'ont poussé, encouragé et soutenu tout au long de ma scolarité pour faire de moi ce que je suis aujourd'hui. Il n'y a pas de mot pour leur décrire ma gratitude, et j'espère que ce témoignage suffira à leur exprimer ne serait-ce qu'une infime partie de ma reconnaissance.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction Générale

La modélisation, ainsi que la simulation à événements discrets s'orientent de plus en plus vers le formalisme DEVS [Zei76], inventé par Bernard P. Zeigler dans les années 1970. Depuis lors, une large communauté scientifique s'est créée autour de ce formalisme à travers le monde, tant dans le but de l'améliorer que d'en étendre l'utilisation à divers domaines applicatifs. Au cours des décennies qui suivirent, de nombreuses extensions virent le jour à l'image de PDEVS [CZ94], Cell-DEVS [WG01], DS-DEVS [Bar98], ... pour ne citer qu'eux. En France, la communauté scientifique se rassemble autour du réseau RED (pour *"Réseau DEVS"*) lors de sa création en septembre 2014. Ce réseau, financé par l'INRA (Institut National de la Recherche Agronomique), a pour objectif de développer, fédérer et promouvoir les travaux orientés sur la théorie de la modélisation et de la simulation autour du formalisme DEVS et de ses extensions.

Les travaux présentés dans cette thèse s'inscrivent dans une démarche d'optimisation des temps de simulation. En effet, face à l'avènement de systèmes de plus en plus complexes, la modélisation, la simulation et l'analyse de ces derniers deviennent de plus en plus coûteux en temps et en mémoire. Même si la multi-modélisation offre une réponse à l'augmentation du besoin en terme de modèles couplés hétérogènes, elle ne suffit pas à garantir une minimisation des temps de calcul. Pour cela, il est nécessaire d'exploiter au maximum l'ensemble des architectures matérielles disponibles, par le biais de la simulation distribuée.

En terminologie DEVS, le modèle global se compose d'un graphe de modèles couplés. Après une mise à plat de la hiérarchie, le modèle global ne contient qu'un unique réseau de modèles atomiques, appelé graphe de modèles. Pour garantir une simulation distribuée optimale en temps, nous proposons une démarche de restructuration de la hiérarchie, en partitionnant ce graphe de modèles. Le partitionnement de ce dernier permet de créer des sous-ensembles de modèles atomiques à distribuer sur les différents nœuds de calcul disponibles. L'optimalité de la distribution repose sur un équilibre des temps d'exécution des modèles des nœuds de calcul et sur la minimisation des échanges d'événements entre les nœuds. L'analyse de la problématique de partitionnement, ainsi que la mise en place de méthodes permettant de la résoudre sont les briques fondamentales permettant de distribuer efficacement une simulation.

Pour garantir un partitionnement optimal, et par conséquent une distribution optimale, il est nécessaire de pondérer le graphe de modèles de façon à ce que ce dernier reflète intégralement la réalité de la simulation. Cette pondération passe par un apprentissage des dynamiques individuelles qui composent le graphe de modèles. En effet, en fonction de la dynamique qui les animent, les modèles atomiques ne vont pas tous émettre la même quantité de messages, ni à la même fré-

quence. Ces variations sont à prendre en compte lors du partitionnement via la pondération des arcs. Dans le cadre de cette thèse, nous proposons une démarche d'apprentissage des dynamiques à l'aide des chaînes de Markov cachées. Les particularités liées au formalisme DEVS doivent être prises en compte lors de chaque apprentissage pour garantir une pondération de qualité. La pondération du graphe de modèles, son partitionnement et la restructuration de la hiérarchie forment une chaîne permettant d'aboutir à la réduction des temps de simulation distribuée. Ils forment les maillons d'une chaîne donnant accès à l'optimisation des simulations distribuées. Cependant, le gain de temps est dépendant du type de la simulation, c'est à dire de la nature de ces modèles atomiques. Nous verrons dans ces travaux, qu'il est extrêmement difficile d'apporter du gain lorsque les modèles atomiques sont asynchrones.

L'optimisation des simulations via la distribution permet de réduire les temps, et permet également de manipuler des simulations beaucoup plus volumineuses en nombre de modèles et en mémoire. Cette avancée contribue à l'évolution des simulations agronomiques à l'image de la prolifération du mildiou de la pomme de terre. Dans un cadre applicatif, nous proposons notre vision de la modélisation du problème "mildiou" à partir des travaux existants en agronomie. Cette thèse ne cherche pas à révolutionner la gestion du risque mildiou, mais seulement à offrir un outil fiable tirant profit de la technologie des objets connectés et permettant de manipuler des échelles spatiales beaucoup plus importantes.

Ce document, tel qu'il est présenté, s'organise de la façon suivante :

— **Chapitre 1 :** État de l'art sur l'état d'esprit *"modélisation et simulation"*, sur le formalisme DEVS dans sa globalité et sur les outils existants de partitionnement de graphe.

— **Chapitre 2 :** Présentation de notre approche de restructuration de la hiérarchie et de l'ensemble des améliorations apportées aux méthodes de partitionnement de notre choix, extraites de la littérature.

— **Chapitre 3 :** Présentation de la démarche de pondération du graphe de modèles par apprentissage des dynamiques individuelles, basée sur les chaînes de Markov cachées.

— **Chapitre 4 :** Présentation du noyau de simulation utilisé, ainsi que l'ensemble des résultats obtenus au cours de cette thèse, dans un contexte parallèle et distribué.

— **Chapitre 5 :** Présentation du Mildiou P. infestant et de son hôte : modélisation de la propagation de cette maladie et de ces effets sur les cultures. Présentation de simulations fictives à partir de données réelles, pouvant hypothétiquement avoir lieu sur la région Hauts de France, plus précisément localisé dans la région de Calais.

Chapitre 1

État de l'art

La simulation et plus précisement, dans le cadre de ce manuscrit, la simulation à événements discrets est régie par un ensemble de règles et de propriétés. Il est important d'en faire un état des lieux afin de les respecter. Ce respect sera le garant d'une mise en oeuvre de qualité. La qualité d'une simulation est aussi fortement dépendante du formalisme utilisé. DEVS (*Discrete Event Specification*) offre un niveau de rigueur et de flexibilité qui en font le candidat idéal. C'est pour cette raison que notre choix s'est porté sur ce formalisme de haut niveau. Ce chapitre a pour objectif, dans un premier temps, de définir formellement ce que sont les concepts de système, de système complexe, de modélisation et de simulation. Dans un second temps, nous introduirons et présenterons le formalisme DEVS dans sa globalité. Un lien avec la simulation distribuée sera établi. Nous verrons par la suite qu'il existe un lien fort entre le formalisme DEVS et la théorie des graphes. La dernière partie de ce chapitre présente des notions fondamentales sur la théorie des graphes ainsi que sur le partitionnement de graphe en général.

1.1 Définitions orientées autour des concepts de simulation et de modélisation

Les domaines de la simulation et de la modélisation sont décrits et définis par un ensemble de termes. De nombreuses définitions existent, nous allons tenter d'en faire une synthèse. Le rôle de cette section est de définir formellement l'ensemble des concepts et des notions en lien plus ou moins étroit avec la simulation et la modélisation. Ces définitions permettent de fixer les idées et de préciser notre point de vue sur les concepts de l'activité de modélisation et de simulation.

1.1.1 Notion de système

Cette sous-section présente la notion de système, ainsi que ses différents niveaux de complexité. Le système étant la brique fondamentale de la simulation, il est important de bien comprendre ce qu'il est et comment le définir. Pour cela, nous proposons de répondre aux questions posées dans cette sous-section.

Qu'est-ce qu'un système ?

La notion de système est à associer aux objectifs de l'étude que l'on mène. En effet, la réalisation d'une étude passe par une phase de questionnement. Or, c'est celle-ci qui définit ce qu'est le système.

> **Définition 1 (Système) :** Un système correspond à l'ensemble des éléments concernés par l'interrogation du modélisateur sur un sujet d'étude. Dès lors que l'on s'intéresse à chaque chose pour en comprendre le fonctionnement (la dynamique), on doit être capable d'identifier les éléments qui constituent l'objet d'étude.

Cette définition intègre l'idée qu'il est important de se limiter aux éléments essentiels et de ne prendre en compte que les éléments susceptibles de répondre à la question posée. Cette étape complexe est généralement réalisée par tâtonnements. Il est aussi important de bien définir la question que l'on se pose sur le système à étudier.

Qu'est-ce qu'un système complexe ?

La définition précédente nous dit qu'un système est un ensemble d'éléments ou d'entités. Ces entités possèdent des comportements ou des dynamiques propres qui par composition et interactions entre elles forment la dynamique du système. Si on étend cette définition, on peut définir la notion de système complexe.

> **Définition 2 (Système complexe) :** Un système complexe est un ensemble composé d'un grand nombre d'entités en interaction locale et simultanée. Le comportement global d'un système complexe émerge de l'interaction des entités qui le compose. La connaissance des propriétés et du comportement des ces éléments isolés n'est pas suffisant pour prédire le comportement global d'un système.

En systémique, on dit qu'une des caractéristiques des systèmes complexes est en effet d'être hiérarchisées et que chaque système est composé de sous-systèmes interconnectés et élément du super-système. Ce point de vue forme un courant fort chez les modélisateurs. Il en existe bien d'autres (holisme, globalisme, ...) mais ils ne seront pas abordés dans cette thèse.

Qu'est-ce qu'un système dynamique et un système spatio-temporel ?

La définition générale d'un système ne fait pas apparaître explicitement la notion de temps et encore moins la notion d'espace. La théorie des systèmes distingue deux choses : la structure interne du système en terme d'éléments qui le compose et le comportement. Le comportement peut être vu comme le résultat de la dynamique interne du système. Le système peut alors être représenté comme une boîte noire composée uniquement d'entrées et de sorties. On ne s'intéresse pas à la compréhension de la dynamique interne. On observe seulement les manifestations externes. Si le temps n'est pas exprimé alors les sorties sont mises en correspondance avec les entrées. Si on place X en entrée alors on a Y en sortie. On dispose alors d'une table de correspondance comme dans le cas d'un circuit logique.

Définition 3 (Système dynamique) : Un système dynamique est un système où la dynamique est définie par rapport à une base de temps, un état courant et une fonction d'évolution des états en fonction des entrées du système.

Pour les systèmes dynamiques, le temps est une composante forte. Il est à l'origine des changements d'états du système. Grâce à cela, il est possible d'identifier l'évolution de l'état du système sur un axe temporel. L'évolution du système se fait à l'aide d'une fonction d'évolution qui porte le nom de fonction de transition.

Définition 4 (Système spatio-temporel) : Un système spatio-temporel est un système dynamique dans lequel les éléments et les processus sont définis relativement à un espace.

Dans ce contexte, l'espace correspond au domaine dans lequel des entités peuvent évoluer. Ces entités possèdent alors des coordonnées relatives à un repère absolu. De même, les processus sont discrets dans l'espace des entités. Les écosystèmes sont de parfaits exemples de systèmes spatio-temporels.

1.1.2 Notion de modèle

La notion de modèle est fondamentale en simulation et en modélisation, il est donc important de bien définir ce qu'est un modèle et quelles en sont les niveaux de spécification.

Qu'est-ce qu'un modèle ?

L'étude d'un système repose sur son observation. C'est à partir de celle-ci qu'est construit le modèle du système, même si cette construction est généralement inconsciente. Nos connaissances et les outils utilisés pour l'observation impliquent de fait un filtrage de la réalité par un modèle. Lorsque l'on regarde une chaise, notre œil en capte une image que l'on considère comme la réalité. Cette réalité est une construction physico-chimique d'une image mentale d'un objet réel. Une caméra infra-rouge et une mouche auront sûrement une autre vision de la même chaise.

Définition 5 (Modèle) : Un modèle est une représentation d'un phénomène à l'aide d'un système qui possède des propriétés analogues à ce phénomène. Celui-ci permet de modéliser son fonctionnement à l'aide d'un ensemble d'équations mathématiques et de programmes informatiques.

Un modèle peut être perçu comme un filtre conditionné par nos connaissances, nos vérités et nos capteurs. Si on restreint le discours à l'activité de modélisation, le modèle est le résultat de l'activité de modélisation. Cette dernière implique un schéma expérimental, un ou des paradigmes et une méthodologie. La notion de paradigme se substitue ici à la notion de vérité dans le cadre général (voir figure 1.1).

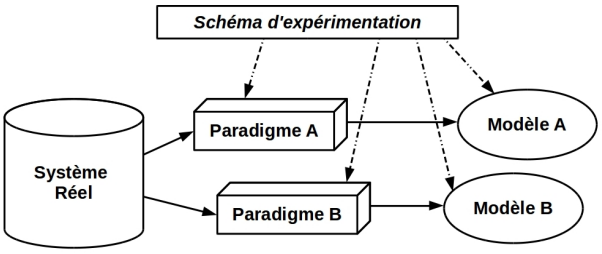


FIGURE 1.1 – L'activité de modélisation

Les niveaux de spécifications formelles des modèles

Intéressons nous à la proposition que fait B. Zeigler, dans [Zei73], à ce sujet. Un système peut être spécifié formellement avec un certain niveau de description. Cette approche purement formelle permet de décrire un système sans aucune hypothèse liée à un paradigme ou à un formalisme.

Le premier niveau de description d'un système diffère peu de la définition de boîtes noires telle que nous l'avons définie précédemment. Les entrées du système sont liées à un intervalle de temps que l'on appelle segment et se note ω. On définit aussi la notion de segment de sortie ρ ou de trajectoire de sortie qui met en relation un intervalle de temps et la sortie observée. B. Zeigler nomme ce type de spécifications des schémas d'observations. Cependant, aucune relation n'est établie entre les entrées et les sorties. Nous sommes ici en présence d'une simple observation d'un phénomène que l'on ne cherche en aucun cas à interpréter.

Si on établit une relation entre les segments d'entrée et les segments de sortie alors on définit le deuxième niveau de spécification formelle d'un système. Ce niveau est appelé observation de la relation d'entrée-sortie. Cette relation a pour but de construire un ensemble de couples (ω, ρ) sachant que ω et ρ sont observés sur le même intervalle de temps. L'inconvénient majeur de cette spécification est d'être non-déterministe. En effet, pour un même segment d'entrée ω, il peut exister plusieurs segments de sortie ρ. Il n'est donc pas possible de prévoir la sortie par une simple observation de l'entrée du système. Les deux premiers niveaux de spécification ne permettent pas la gestion des systèmes dynamiques. Pour remédier à cela, il est nécessaire de définir un troisième niveau de spécification en ajoutant la notion d'état. L'ajout de la spécification de l'état initial du système permet de rendre déterministe la relation d'entrée-sortie. On parle alors d'observation de la fonction d'entrée-sortie.

Le quatrième niveau d'abstraction s'attache à définir la spécification formelle d'un système dynamique. A ce niveau, on va chercher à spécifier le comportement interne du système. Pour cela, on introduit la notion d'état interne et de fonctions de transition.

$$S = < T, X, Y, \Omega, Q, \Delta, \Lambda >$$

où :

- T est la base de temps
- X est l'ensemble des valeurs d'entrées possibles

- Y est l'ensemble des valeurs de sorties possibles

- Ω est l'ensemble des segments d'entrées admissibles

- Q est l'ensemble des états du système

- $\Delta : Q \times \Omega \rightarrow Q$ est la fonction de transition

-$\Lambda : Q \rightarrow Y$ est la fonction de sortie

Cette formalisation des systèmes dynamiques permet de décrire le comportement du système en fonction de son état interne et des événements d'entrée. Or, dans cette vision, on considère que le système n'est pas autonome et qu'il n'a donc pas de dynamique propre puisque seuls des événements externes peuvent modifier l'état interne du système.

Le cinquième et dernier niveau introduit le concept de couplage de modèles. Le modèle global est alors un ensemble de modèles interconnectés. Ces connexions représentent les relations entre les modèles et les événements échangés entre les modèles. La figure 1.2 illustre cette notion de couplage. A ce niveau, on considère que le modélisateur connaît la structure interne de son système et qu'il est capable de le décomposer en sous-modèles.

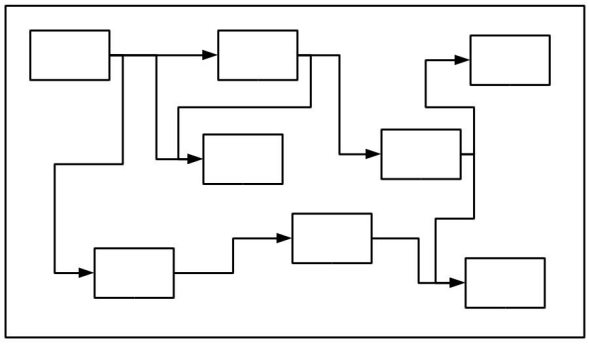


FIGURE 1.2 – Exemple de couplage de modèles

1.1.3 Notion de paradigme et de formalisme

Cette sous-section définit formellement ce que sont les notions de paradigme et de formalisme, au travers de questionnements.

Qu'est-ce qu'un paradigme ?

Sans le préciser lorsque B. Zeigler propose ses niveaux de spécifications, il exploite un paradigme celui de la modélisation à événements discrets.

Définition 6 (Paradigme) : Un paradigme est un cadre de pensée composé par un ensemble d'hypothèses fondamentales, de lois et de moyens à partir desquels les modèles peuvent se développer.

Il existe un grand nombre de paradigmes comme par exemple : le paradigme multi-agents, les équations différentielles ou encore les événements discrets. Dans la plupart des cas, la notion

de paradigme est difficilement dissociable de la notion de formalisme. En effet, les équations différentielles sont à la fois un paradigme et un outil de formalisme mathématique.

En règle générale, le paradigme est utilisé au niveau sémantique tandis que le formalisme est l'outil opérationnel du paradigme. À un paradigme est associé un ou plusieurs formalismes. De plus, on peut concevoir des modèles en s'appuyant sur les concepts d'un paradigme et spécifier le modèle à l'aide d'un formalisme qui n'est pas obligatoirement issu du paradigme.

Qu'est-ce qu'un formalisme ?

Les définitions précédentes ont posé les bases de la modélisation en définissant ce qu'est un modèle. Nous avons vu aussi qu'il était possible de formaliser un système selon différents niveaux d'abstraction, niveau qui dépend la plupart du temps du niveau de connaissances sur le système.

Revenons, par exemple, sur la notion de fonction de transition. Cette fonction a pour objectif de déterminer le nouvel état du système en fonction des événements d'entrée (ou externes). La question qui reste en suspens est comment spécifier cette fonction.

> **Définition 7 (Formalisme) :** Le formalisme est l'outil permettant de spécifier l'ensemble des fonctions nécessaires à l'évolution du modèle. Il est l'outil d'expression des paradigmes.

On peut utiliser, par exemple, les automates à états finis pour spécifier les changements d'états mais ce n'est pas la seule possibilité.

1.2 Le formalisme DEVS

DEVS, ou *Discrete Event System Specification*, est un formalisme à événements discrets inventé par Bernard P. Zeigler et présenté pour la première fois dans la première version de son ouvrage [Zei76] en 1976. DEVS est un formalisme de haut niveau basé sur les événements discret pour la modélisation de systèmes complexes et continus. Son utilisation ne se limite pas seulement à la modélisation, il permet également de faire de la simulation et de l'analyse de systèmes complexes. Ce formalisme constitue un support mathématique efficace et solide pour la modélisation des systèmes à événements discrets. Il est notamment reconnu pour son expressivité, son abstraction, son universalité et son indépendance de toute implémentation.

1.2.1 Les outils fondamentaux du formalisme DEVS

DEVS intègre la dimension temporelle de façon explicite et permet de manipuler conjointement les deux concepts d'état et d'événement. Pour cela, il s'oriente vers deux objets élémentaires : les modèles atomiques et les modèles couplés.

Modèles atomiques

Un modèle atomique est conçu pour gérer la dynamique du système à simuler. Il se définit par un ensemble de ports d'entrée, de sortie, et un ensemble de fonctions. Ces informations sont

concaténées dans le 7-uplet suivant :

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

Où,

- X est ensemble des ports et des valeurs d'entrée.

- Y est ensemble des ports et des valeurs de sortie.

- S est ensemble des états partiels ou séquentiels incluant les variables d'états du système.

- $\delta_{int} : S \to S$ est la fonction de transition interne dont le rôle est de faire passer le système d'un état à un autre de manière autonome. Elle spécifie les états futurs des états actifs, à condition que leur durée de vie ne soit pas infinie.

- $\delta_{ext} : Q \times X \to S$ est la fonction de transition externe dont le rôle est de faire passer le système d'un état à un autre suite à l'apparition d'un événement externe perturbateur

  Où, $Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$ est l'ensemble des états totaux du système et e représente le temps écoulé dans l'état s depuis la dernière transition. Le concept d'état total (s,e) permet de spécifier un état futur en fonction du temps écoulé dans l'état présent.

- $\lambda : S \to Y$ est la fonction de sortie. Elle n'est activée que lorsque le temps écoulé dans un état donné est égal à sa durée de vie.

- $ta : S \to \mathbb{R}_0^+ \cup +\infty$ est la fonction d'avancement du temps, son rôle est de donner la durée pendant laquelle le système sera dans un certain état.

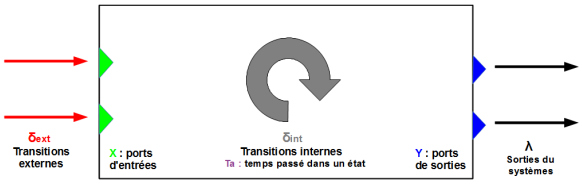


FIGURE 1.3 – Schématisation d'un modèle atomique

La figure 1.3, illustre le fonctionnement d'un modèle atomique. Celle-ci se compose d'un bloc représentant le système M (ou le modèle), ainsi que des triangles qui représentent les différents ports X, Y, vert pour les entrées et bleu pour les sorties. Les différentes transitions $\delta_{int}, \delta_{ext}$ ainsi que les sorties sont représentées par des flèches. Les rouges symbolisent l'arrivée d'un événement "perturbateur" sur un ou plusieurs ports d'entrées ce qui est à l'origine du déclenchement de la fonction de transition externe pour modifier l'état courant. La flèche grise symbolise le calcul de la fonction de transition interne qui a lieu lorsqu'un état s_k atteint la fin de sa durée de vie à $t + t_a(s_k)$ sans être perturbé par un événement extérieur. À chaque transition interne est associée la

fonction de sortie λ, symbolisée par les flèches noires. Cette fonction émet les sorties du système par le biais des ports de sortie.

Un système qui ne reçoit aucun événement extérieur est dit *autonome*. La dynamique du système est donc exclusivement gérée par la fonction de transition interne. Celle-ci est définie pour spécifier les changements d'états qui sont liés exclusivement à l'état interne et au temps. Le rôle de la fonction $t_a(s_k)$ est de donner le temps durant lequel le système restera dans l'état s_k, si cette durée est nulle l'état s_k s'appellera *état transitoire* et si au contraire cette durée est infinie il s'appellera *état passif*. Pour $ta(s) = \infty$, cela signifie que le système reste indéfiniment dans l'état courant tant qu'aucun événement externe ne se produit. Et au contraire, pour $ta(s) = 0$ le système déclenche instantanément $\lambda(s)$ et effectue une transition vers l'état $s' = \delta_{int}(s)$. Cette dernière possibilité est accessible via l'extension Parallel-DEVS.

Modèles couplés

Il est possible de coupler plusieurs modèles atomiques dans le but de former une super structure, appelée modèle couplé. Un modèle couplé possède la même structure qu'un modèle atomique, ainsi que les mêmes propriétés. Chaque modèle couplé définit l'ensemble des connexions qui lient les modèles qui le compose. Celui-ci peut se composer à la fois de modèles atomiques et de modèles couplés qui peuvent eux même se composer de modèles couplés et ainsi de suite. Cette composition donne naissance à une hiérarchie de modèles propre au formalisme DEVS, cette hiérarchie est présentée en section 1.6. Formellement, un modèle couplé est défini par :

$$N = \langle X_N, Y_N, D, \{M_d, d \in D\}, EIC, EOC, IC, Select \rangle$$

Où,

- X_N et Y_N ont la même définition que pour les modèles atomiques.
- D est l'ensemble des identifiants des modèles composants le modèle couplé.
- M_d est l'ensemble des modèles (atomiques ou couplés) composants le modèle couplé.
- $EIC = \{((N,a),(d,b)) \mid a \in IPorts,\ b \in IPorts_d\}$ est la liste des ports d'entrée du modèle couplé connectés aux ports d'entrée des sous-modèles

  où $IPorts$ est l'ensemble des ports d'entrées de N

  et $IPorts_d$ est l'ensemble des ports d'entrées des sous modèles $d \in D$
- $EOC = \{((N,a),(d,b)) \mid a \in OPorts,\ b \in OPorts_d\}$ est la liste des ports de sorties du modèle couplé connectés aux ports de sorties des sous-modèles.

  où $OPorts$ est l'ensemble des ports de sorties de N

  et $OPorts_d$ est l'ensemble des ports de sorties des sous modèles $d \in D$
- $IC = \{((i,a),(j,b)) \mid i,j \in D,\ i \neq j,\ a \in OPorts_i,\ b \in OPorts_j\}$ est la liste des connexions internes au modèle couplé (connexion entre les sous-modèles). Une sortie d'un modèle ne

peut pas être couplée à l'une de ses entrées.

- *Select* définit une priorité entre événements simultanés destinés à des composants différents

La figure 1.4 fournit une représentation graphique d'un modèle nommé **ALPHA** qui se compose lui-même de deux modèles couplés **alpha_i**, **alpha_ii** et d'un modèle atomique F. Chacun des modèles couplés se compose lui-même de plusieurs modèles atomiques. Afin de fournir un exemple simple, le sous-modèle couplé **alpha_ii** est développé. Ses caractéristiques sont :

$$
\begin{aligned}
X_N &= \{in\} \text{ ainsi que l'ensemble des valeurs prises par } in \\
Y_N &= \{out\} \text{ ainsi que l'ensemble des valeurs prises par } out \\
M &= \text{deux modèles atomiques} \\
D &= \{D, E\} \\
EIC &= \{((\textbf{alpha\_ii}, in), (D, in_d))\} \\
EOC &= \{((E, out_e), (\textbf{alpha\_ii}, out))\} \\
IC &= \{((D, out_d), (E, in_e))\}
\end{aligned}
$$

Les ensembles EIC, EOC et IC ont pour rôle de gérer le transfert des événements entre les différents modèles. Cet exemple montre également la nature hiérarchique des modèles couplés.

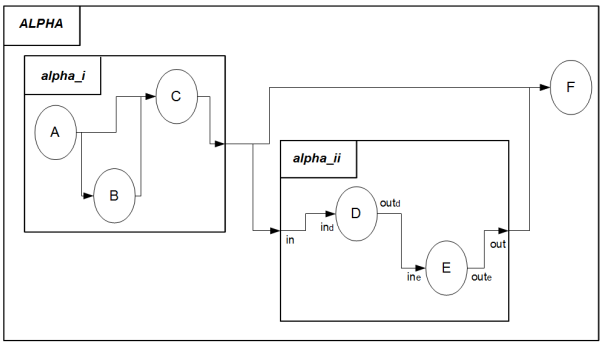


FIGURE 1.4 – Exemple et représentation d'un modèle couplé

Il a été prouvé que DEVS est « fermé sous couplage », ce qui signifie qu'un modèle couplé (quel que soit le nombre de ses sous modèles) peut être transformé en modèle atomique unique, qui en est l'exact équivalent. Cette propriété a le grand avantage de faciliter la construction hiérarchique des modèles par l'application récursive des procédures de couplage [Zei84]. Cette propriété est fondamentale pour la suite des travaux menés dans cette thèse.

Exemple d'exécution d'un modèle

La figure 1.5 illustre l'évolution d'un modèle DEVS sur un exemple. Le système se compose de 5 états discrets. Au temps $t = 0$, le système se trouve dans l'état s_0. Au temps $t_1 = t + t_a(s_0)$, le système n'ayant pas été perturbé par un événement extérieur, la fonction de sortie est activée et Y se voit attribuer la valeur produite par la fonction de sortie $\lambda(s_0)$. Après avoir affecté les ports de

sortie, la fonction de transition interne $\delta_{int}(s_0)$ est appelée dans le but de faire basculer le système dans l'état $s_1 = \delta_{int}(s_0)$. L'état s_1, initialement prévu pour un changement d'état à la date $t + t_a(s_1)$, se voit perturbé par une entrée au temps $e = t_2 < t + t_a(s_1)$. La fonction de transition externe est donc appelée pour déterminer le nouvel état. La fonction de sortie n'est pas appelée car elle ne l'est que lorsque $e = t_a$, son appel précède exclusivement celle de la fonction de transition interne. Au temps t_2, le système bascule dans l'état $s_3 = \delta_{ext}(s_1, e, x)$. Si aucun événement perturbateur n'avait eu lieu, le système aurait basculé dans l'état $s_2 = \delta_{int}(s_1)$ au temps $t_2 = t + t_a(s_1)$.

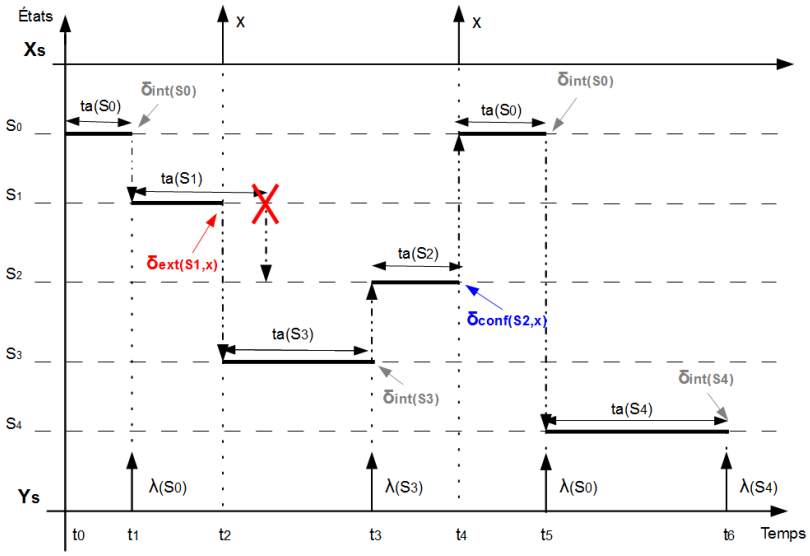


FIGURE 1.5 – Exemple de graphe de transition

Le système évolue par transition interne jusqu'au temps t_4. À cet instant, le système est soumis à un conflit : un événement est déposé sur les ports d'entrée au même moment où la fonction de transition interne est censée être appelée. Le formalisme DEVS, initialement conçu, ne permettait pas de gérer ce genre de problème. Cependant, l'extension Parallel-DEVS présenté en section 1.2.2 offre une solution : la fonction de confluence δ_{conf}. Cette fonction règle le conflit en indiquant le nouvel état mais sans prise de valeur de la part de la fonction de sortie. Pour terminer, le système atteint son état final s_4 au temps t_6 par transition interne.

En résumé, lorsque le système reste dans un état s_k durant une période $t_a(s_k)$ sans être perturbé par un événement extérieur, la fonction λ est appelée pour générer une sortie avant application de la fonction de transition interne dont le rôle est de changer l'état du système de manière autonome. Dans le cas contraire, la fonction de transition externe est appelée pour déterminer le nouvel état et aucune sortie n'est produite.

1.2.2 L'extension Parallel-DEVS

Le formalisme Parallel-DEVS (PDEVS) [CZ94, Cho96], est une extension du formalisme DEVS qui introduit le concept d'événements simultanés. Ce formalisme permet une plus grande souplesse au niveau de la gestion des priorités en les traitant non plus au niveau du modèle couplé mais au niveau du modèle atomique. PDEVS, ajoute dans le modèle atomique une fonction de

confluence :

$$\delta_{con}(s,\emptyset) = \delta_{int}(s)$$

Cette fonction permet de gérer les conflits entre des événements programmés simultanément. Contrairement à DEVS, ce formalisme ne traite plus seulement un événement d'entrée et de sortie à la fois, mais un ensemble d'événements, pouvant se produire au même moment. Ces ensembles d'événements portent le nom de "bags" et se note X^b et Y^b. La fonction d'un "bag" est de regrouper les événements actifs à une même date et de traiter leurs effets.

Pour éviter les redites, seuls les paramètres ayant subit un changement lors du passage à PDEVS sont présentés. PDEVS définit un modèle atomique par :

$$M_p = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

Où,

$\delta_{ext} : Q \times X^b \to S$ est la fonction de transition externe.

$\delta_{con} : S \times X^b \to S$ est la fonction de confluence dont le rôle est de gérer le cas où plusieurs événements se réalisent à la même date dans le but de spécifier l'état futur qui en résulte.

Un modèle PDEVS se comporte comme suit : si aucun événement externe n'apparaît, le système reste dans l'état s durant $ta(s)$ pas de temps. Lorsque $e = ta(s)$, le système change d'état par appel de la fonction δ_{int}. Si un événement externe de valeur x apparaît, lorsque le système est dans l'état (s,e), le système change d'état par appel de la fonction $\delta_{ext}(s,e,x)$. Si il apparaît quand $e = ta(s)$, le système change son état par appel de la fonction de conflit $\delta_{con}(s,x)$. La fonction de conflit définie par défaut est :

$$\delta_{con}(s,x) = \delta_{ext}(\delta_{int}(s),0,x)$$

Mais le modélisateur peut préférer l'ordre inverse :

$$\delta_{con}(s,x) = \delta_{int}(\delta_{ext}(s,ta(s),x))$$

Il est tout à fait possible de définir sa propre fonction de conflit en cas de besoin.

La formalisation d'un modèle couplé subit quelques modifications en PDEVS :

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\} \rangle$$

Où,

$\forall d \in D$, M_d est un modèle PDEVS

$\forall d \in D \cup \{N\}$, I_d est l'ensemble d'influence de d :

$I_d \subseteq D \cup \{N\}$

$d \notin I_d$, $\forall d \in D \cup \{N\}$, $\forall i \in I_d$, $Z_{i,d}$ est une fonction,

le transfert de sortie entre i et d :

$$Z_{i,d} : X \rightarrow X_d, \text{ si } i = N$$

$$Z_{i,d} : Y_i \rightarrow Y, \text{ si } d = N$$

$$Z_{i,d} : Y_i \rightarrow X_d, \text{ si } i \neq N \text{ et } d \neq N$$

L'ensemble d'influence de d est un ensemble de modèles qui interagissent avec d. $Z_{i,d}$ spécifie le type de relations qui existe entre les modèles i et d. En PDEVS, la fonction *Select* est remplacée par la fonction de conflit δ_{con}.

PDEVS est un formalisme opérationnel qui offre des algorithmes pour son exécution. Ces algorithmes sont la base du fonctionnement de la structure PDEVS. Ils sont intégralement présentés dans le livre de Bernard P. Zeigler [ZKP00]. Dans le cadre de cette thèse, nous présentons uniquement les algorithmes des coordinateurs car ils permettent d'exécuter simultanément les éléments qui les composent.

Algorithmes des coordinateurs dans PDEVS

L'approche Parallel-DEVS, pour paralléliser ou distribuer une simulation, se contraint à un respect strict de la causalité. L'algorithme 1 présente l'ensemble des variables nécessaires à la compréhension des algorithmes de gestion des coordinateurs.

Algorithme 1 La liste des variables

$DEVS = (X, Y, D, \{M_d\}, \{i_d\}, \{Z_{i,d}\})$
parent : coordinateur parent
tl, tn : date de dernière activation et de prochaine activation
$event - list$: liste des événements traités lors de la transition externe - bag
IMM : ensemble des modèles dont la date d'activation est imminente
mail : ensemble des événements externes
y_{parent} : sorties à destination du coordinateur parent si les événements de sorties n'ont pas de destinataire au sein du modèle couplé géré par le coordinateur
$\{y_d\}$: ensemble des sorties des sous-modèles d gérées par le coordinateur

Dans une structure hiérarchique, chaque coordinateur gère un échéancier et achemine les messages vers et entre ses enfants. L'échéancier stocke les événements internes (un événement interne par enfant) triés par ordre de réveil des modèles. Ces temps sont produits par la fonction t_a pour chaque modèle atomique et à partir de la date de simulation courante. À chaque itération, les coordinateurs créent un ensemble de messages immédiats correspondant à tous les événements possédant la même date de réveil (IMM).

Algorithme 2 Algorithme de traitement des i-messages au temps t

Pour chaque $d \in D$ **faire**
  envoyer en parallèle des i-messages aux enfants de d
Fin pour
trier de la *liste d'événements* suivant les tn_d
$tl = max\{tl_d | d \in D\}$
$tn = min\{tn_d | d \in D\}$

Algorithme 3 Algorithme de traitement des y-messages au temps t, avec des sorties y_d provenant de d

Si ce n'est pas le dernier $d \in IMM$ **Alors**
  ajouter (y_d, d) à *mail*
  marquer d comme traité
Sinon
  Si c'est le dernier $d \in IMM$ **Alors**
    $y_{parent} = \emptyset$
  Finsi
Finsi
destinataires = $\{r | r \in D, N \in I_r, Z_{N,r}(x) \neq \emptyset\}$
Pour chaque $d \in I_N \wedge d$ traité **faire**
  Si $Z_{d,N}(y_d) \neq \emptyset$ **Alors**
    ajouter y_d à y_{parent}
  Finsi
Fin pour
envoyer y-message(y_{parent}, t) à parent
Pour chaque enfant r avec $d \in I_r \wedge d$ traité $\wedge Z_{d,r}(y_d) \neq \emptyset$ **faire**
  Pour chaque $d \in I_r \wedge d$ traité $\wedge Z_{d,r}(y_d) \neq \emptyset$ **faire**
    ajouter $Z_{d,r}(y_d)$ à y_r
  Fin pour
  envoyer x-message(y_r, t) vers r
Fin pour
Pour chaque $r \in IMM \wedge y_r = \emptyset$ **faire**
  envoyer x-message($\emptyset, t$) vers r
Fin pour
trier la *liste d'événements* suivant les tn_d
$tl = t$
$tn = min\{tn_d | d \in D\}$

Algorithme 4 Algorithme de traitement des $*$-messages au temps t

Si $t \neq tn$ **Alors**
  erreur : mauvaise synchronisation
Finsi
$IMM = \{d | (d, tn_d) \in$ (event-list $\wedge tn_d = tn)\}$
Pour chaque $r \in IMM$ **faire**
  envoyer en parallèle les $*$-messages($*$,t) à r
Fin pour

Les messages reçus par un coordinateur sont décrits par les algorithmes 1, 2, 4, 5 et 3. `i-message` est utilisé pour initialiser les enfants. `*-message` est utilisé pour exécuter les sorties des enfants. `x-message` et `y-message` sont utilisés pour router les messages au sein du modèle couplé. En PDEVS, tous les modèles imminents sont autorisés à s'exécuter en même temps, contrairement à DEVS où les modèles imminents sont activés séquentiellement. Les sorties des *IMM* sont collectées dans des ensembles d'événements externes appelés *mail* dans les précédents algorithmes.

Algorithme 5 Algorithme de traitement des x-messages au temps t

Si $\neg(tl \leqslant t \leqslant tn)$ **Alors**
  erreur : mauvaise synchronisation
Finsi
destinataires = $\{r | r \in D, N \in I_r, Z_{N,r}(x) \neq \emptyset\}$
Pour chaque $r \in$ destinataires **faire**
  envoyer en parallèle les x-messages($\emptyset$, t) à r
Fin pour
Pour chaque $r \in IMM \wedge \neg \in$ destinataires **faire**
  envoyer en parallèle les x-messages($\emptyset$, t) à r
Fin pour
trier la *liste d'événements* suivant les tn_d
$tl = t$
$tn = min\{tn_d | d \in D\}$

1.2.3 Hiérarchie structurée d'un modèle DEVS

Parallèlement à l'élaboration des différents modèles DEVS présentés précédemment, B.P. Zeigler a développé le concept de simulateur abstrait [ZKP00]. Un simulateur abstrait représente une description algorithmique permettant de mettre en œuvre les fonctions du modèle, afin de générer son comportement. Il en existe 3 types :

le **coordinateur racine** assure la gestion globale de la simulation en gérant notamment l'horloge globale. Il a la charge d'interroger ses enfants afin de connaître la date du prochain événement.

le **coordinateur** assure le routage des messages entre les modèles couplés et collecte les dates de réveil de chacun de ses enfants. Il est le représentant abstrait du modèle couplé.

le **simulateur** assure la simulation des modèles atomiques en utilisant les fonctions définies par DEVS. C'est le représentant abstrait du modèle atomique.

DEVS offre une représentation opérationnelle d'un modèle couplé en utilisant la notion de simulateur abstrait, celle-ci porte le nom de hiérarchie structurée de simulateurs abstraits, ou plus simplement d'arbre de simulation. L'intérêt d'une telle représentation est de pouvoir observer simplement les liens de dépendances entre les structures abstraites (simulateurs - coordinateurs, coordinateurs - coordinateurs). En revanche, elle ne permet pas de connaître les liens qui unissent les simulateurs entre eux.

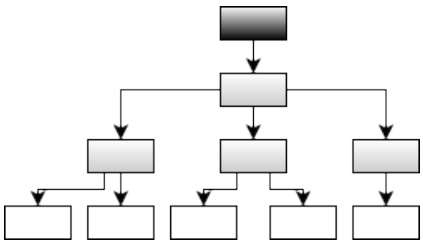


FIGURE 1.6 – Représentation hiérarchique structurée de simulateurs abstraits d'un modèle DEVS

La figure 1.6 présente un exemple d'arbre de simulation. Celle-ci se compose d'un coordinateur racine représenté par un rectangle gris foncé, de coordinateurs représentés par des rectangles gris clairs et de simulateurs représentés par des rectangles blancs. Le coordinateur racine fait évoluer la simulation par réception de la date du prochain événement fourni par son coordinateur enfant. Celui-ci n'existe donc pas dans une représentation classique d'un modèle couplé. Le rôle des coordinateurs est de récupérer les prochaines dates d'activation des simulateurs et de faire remonter l'information au coordinateur racine par transmission successive aux différents coordinateurs parents. Dans cet exemple, les coordinateurs du troisième niveau récupèrent les prochaines dates d'activations relatives aux simulateurs du dernier niveau et transmettent l'information à leur coordinateur parent, c'est à dire le coordinateur du second niveau. L'évolution de la simulation est dépendante de la politique de transmission des informations entre les coordinateurs jusqu'au coordinateur racine. La sous section suivante présente la politique de transmission standard, utilisée dans les simulations DEVS hiérarchiques.

Évolution d'un modèle hiérarchique DEVS

Au temps $t = 0$, le coordinateur racine transmet un message à chaque simulateur, en parcourant les différents niveaux de la hiérarchie, dans le but de déterminer la prochaine date d'activation. À leur tour, les simulateurs retournent leur prochaine date d'activation à leur coordinateur parent. Suite à cela, chaque coordinateur parent sélectionne la plus petite date d'activation et la transmet à son tour à son parent. Ce processus est itéré jusqu'à ce que la plus petite date d'activation soit retournée au coordinateur racine.

Plus formellement, soit t le temps de simulation courant, t_{N_i} la prochaine date d'activation du simulateur i, $(*,t)$ le message transmis par le coordinateur racine aux simulateurs afin d'activer le simulateur dont la date de réveil est t et $(done,t_N)$ le message contenant la date du prochain réveil à transférer au coordinateur parent.

La simulation se déroule en deux phases :

1. Recherche du simulateur dont la prochaine date d'activation est la plus proche
   - chaque simulateur modifie son t_N et le retourne à son coordinateur parent via le message $(done,t_n)$
   - le coordinateur parent sélectionne le plus petit t_N reçu et le transmet à son parent, ainsi de suite jusqu'au coordinateur racine
   - le coordinateur racine avance le temps de simulation t à $min(t_N)$ et commence la phase d'exécution
2. Exécution de la simulation
   - le coordinateur racine transmet le message $(*,t)$ à l'ensemble de la hiérarchie jusqu'à atteindre les simulateurs
   - le simulateur dont le t_N est égal à l'instant de simulation t réalise sa transition et le processus reprend à l'étape 1

1.2.4 La simulation parallèle / distribuée

Il est important de bien distinguer la simulation parallèle de la simulation distribuée [Fuj00]. Comme le montre la figure 1.7, la simulation parallèle implique l'exécution d'une unique simulation sur un ensemble de processeurs étroitement couplés (à mémoire partagée). Alors que la simulation distribuée implique l'exécution d'une simulation unique sur un ensemble de processeurs à couplage lâche (par exemple, des ordinateurs interconnectés par un réseau LAN ou WAN). Une autre forme de simulation existe : la réplication. Dans ce cas, ce n'est qu'une duplication du même modèle de simulation avec des paramètres différents qui est exécuté sur plusieurs nœuds de calcul. Les instances de simulations sont totalement indépendantes. Cette stratégie est utilisée pour les plans d'expérience, par exemple. Ce cas est en dehors de notre étude.

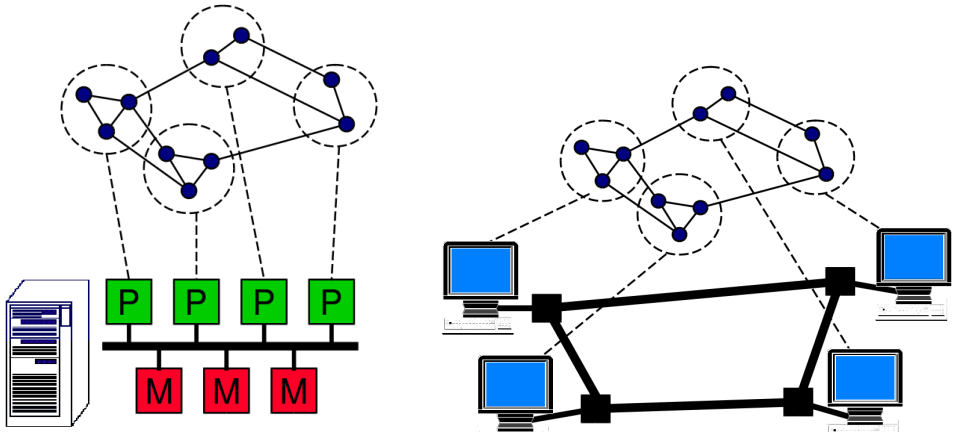


FIGURE 1.7 – Illustration de la simulation parallèle à gauche, et de la simulation distribuée à droite

Ces différentes catégories ont toutes pour objectif de réduire le temps d'exécution des modèles proportionnellement au nombre de processeurs, mais aussi de gérer des simulations plus volumineuses, notamment lorsque la mémoire nécessaire est insuffisante en mono-processeur. Depuis les années 80, la communauté des chercheurs en HPC (*High Performance Computing*) est très active. A la fois, dans les domaines des jeux vidéos distribués et massivement multi-joueurs mais aussi dans le domaine des simulateurs militaires. Depuis 1996, une architecture s'est imposée : HLA (*High Level Architecture* [KLG+02]). Elle repose sur un *runtime* dont le rôle est de gérer à la fois la synchronisation, le transport des données entre les sous-parties d'un simulateur et l'initialisation de la simulation. Chaque simulateur fait ensuite appel à une couche d'interface afin de rendre HLA-compliant le simulateur. Il existe des implémentations dans le monde DEVS (DEVS-HLA [ZL98] [ZCLS98]). Le module "synchronisation temporelle" est naturellement le plus sensible. Nous allons voir les différentes stratégies possibles. HLA, quant à lui, repose sur la notion de *lookhead* (durée au delà de sa date actuelle pendant laquelle un simulateur atteste qu'il n'émettra pas de message) [Fuj87].

La simulation à événements discrets peut être considérée comme une séquence de calculs d'événements. Où, on associe à chaque calcul une date d'activation (time stamp) indiquant le moment où l'événement se produit dans le système. Chacun de ces calculs peut modifier les variables d'états, et (/ou) planifier de nouveaux événements dans le futur. La simulation à événements dis-

crets parallèles peut être considéré comme un ensemble de simulations à événements discrets séquentiels dont l'exécution est effectuée sur différents processeurs, qui communiquent entre eux par l'envoi de message dans le temps. En pratique, chaque envoi de message correspond à la planification d'un événement. Pour garantir une simulation fiable, il est impératif de satisfaire la *contrainte de causalité locale*. Celle-ci impose que les événements de chaque processus logique soient traités dans l'ordre de leur date d'activation. La causalité temporelle impose donc un ordre partiel entre les événements [Lam78] et le respect de cette contrainte suffit à garantir l'obtention d'un résultat équivalent entre simulation parallèle, distribuée et séquentielle.

L'évolution des simulations parallèles et distribuées passe donc par l'utilisation d'algorithmes de synchronisation temporelle (*Time Management*). Ces simulations supposent que l'on considère la simulation comme composée d'un ensemble de *Logical Processors* (processeur logique ou nœud de calcul) qui communiquent en échangeant des messages datés ou estampillés (événements). Chaque LP exécute une simulation à événements discrets locale, qui maintient un échéancier avec les événements locaux et ceux reçus des simulateurs n'appartenant pas au même LP, un état local courant et une horloge locale qui contient la date locale actuelle. Dans un contexte DEVS, un LP est associé à un coordinateur.

Deux politiques de gestion de la synchronisation sont possibles :

— la synchronisation conservative (ou pessimiste) : interdit la violation de la contrainte de causalité locale. Dans ce cas, la sortie d'événements d'un modèle à l'autre est bloquée tant qu'il y a un problème de causalité.

— la synchronisation optimiste : autorise la violation de la contrainte de causalité. Cependant, si son utilisation s'avérait justifiée, il serait nécessaire d'effectuer un retour en arrière, via l'utilisation d'un mécanisme de *roolback* afin de garantir l'exactitude des résultats.

Dans le premier cas, nous parlons d'approche pessimiste (ou sans risque) et dans le second cas, nous parlons d'approche optimiste. Dans le cas pessimiste, un LP sélectionne et traite l'événement (local ou distant) dont la date d'activation est minimale de l'échéancier. Après traitement, un nouvel événement (local ou à destination d'un autre LP) est généré et la date logique est actualisée avec la date de l'événement traité. Lorsqu'un événement dont la date est T est traité, le LP doit s'assurer de ne pas être influencé par un autre LP avec un événement de date T' < T, et par conséquent, respecter la causalité. L'objectif des premiers algorithmes de synchronisation est donc de déterminer si les événements à traiter au sein des LP sont conformes à ce principe de causalité. Ces algorithmes ont été proposés par [Bry77] et [CM79]. Néanmoins, des situations de blocage peuvent apparaître si l'un des LP ne fournit pas d'événement estampillé aux LP auxquels il est connecté. Dans cette situation, les LP récepteurs ne peuvent pas décider. La notion de *null-message* est alors introduite pour pallier à ce problème. Ce message particulier contient la date de prochaine activation. Cette solution a un coût puisque tous les LP n'émettant pas d'événements doivent envoyer un *null-message* à l'ensemble des LP influencés. Avec [CM81], une amélioration est apportée en activant le mécanisme de *null-message* seulement lors d'un problème de blocage. En résumé, la performance de la simulation parallèle ou distribuée est donc très liée au choix de la durée pendant laquelle les LPs n'émettront pas de message. Cette durée, appelée *Lookahead*, est totalement dépendante du modèle à simuler.

Dans le cas optimiste, la violation de la contrainte de causalité est possible. Le LP traite les événements locaux ou distants déjà reçus sans se préoccuper des potentiels nouveaux événements et peut donc conduire à l'omission de certains événements externes et au non respect de la causalité [Sam85]. Si la causalité n'est pas respectée par l'arrivée d'un événement dont la date d'activation est supérieure à la date du LP), la simulation doit revenir à son état passé conformément à la date de l'événement reçu. Ce retour en arrière peut se propager car si le LP a émis des événements, ces derniers doivent aussi être annulés. Un mécanisme de *Rollback* est alors activé [Jef85]. Jefferson propose l'algorithme TimeWrap avec l'idée suivante : on sait que l'état global de la simulation est valide jusqu'à la date de l'événement non traité dont la date est la plus éloignée dans le passé pour l'ensemble de la simulation. L'algorithme Time Warp définit donc la date du message non traité le plus ancien comme la date minimale globale (GVT - *Global Virtual Time*). Cette date est donc la date minimale jusqu'à laquelle la simulation pourra revenir. Chaque LP doit donc mémoriser l'ensemble des états et des événements générés jusqu'à cette date. Au fur et à mesure que GVT avance, ces informations sont supprimées. Naturellement, le coût de ce type d'algorithme peut être très important, tant en terme de mémoire pour le stockage des informations qu'en terme de fréquence de *roolback*.

1.2.5 Structure à plat d'un modèle DEVS

Dans cette section, nous introduisons le concept de mise à plat de la hiérarchie de modèles. Celle-ci se constitue d'un unique coordinateur gérant les événements de toute la hiérarchie afin d'éviter le routage des événements. Cette démarche permet de réduire le surcoût lié à la gestion des événements.

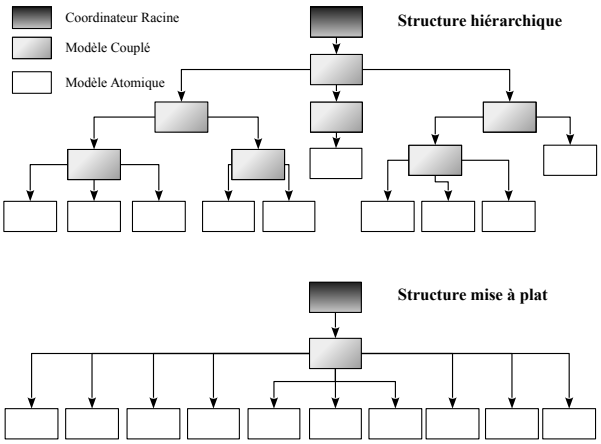


FIGURE 1.8 – Exemple de mise à plat d'une hiérarchique structurée de simulateurs abstraits d'un modèle DEVS

Le principe est simple, la mise à plat de l'arbre de simulation simplifie la hiérarchie en supprimant tous les modèles couplés de la hiérarchie de modèles, à l'exception du premier. Comme pour les simulateurs abstraits, un coordinateur est attaché au modèle couplé et à chaque modèle atomique est attaché un simulateur. Si le modèle ne possède qu'un seul modèle atomique, le simulateur fonctionne alors de la même manière que les simulateurs abstraits, le coordinateur racine,

étant connecté à un simulateur qui traite les événements du modèle atomique. La mise à plat de la hiérarchie est possible grâce aux propriétés de fermeture sous couplage énoncées dans la section 1.2.1. Cette propriété justifie que la mise à plat de la hiérarchie n'ait pas d'impact sur le fonctionnement des simulateurs, ni sur le déroulement de la simulation à condition que celle-ci soit bien gérée par le noyau de simulation.

La figure 1.8 présente un exemple de mise à plat d'une structure hiérarchique DEVS. Suite à la mise à plat, nous pouvons constater que l'arbre de simulation se compose d'un unique coordinateur gérant l'intégralité des simulateurs. Cette structure a l'avantage de réduire considérablement la propagation de message au cours de l'exécution. Ce type de structure permet également d'accéder facilement à une autre représentation de la simulation. Celle-ci, contrairement à l'arbre de simulation, fournit exclusivement la dépendance entre les simulateurs et donne ainsi une vision globale de leurs communications. Il s'agit d'un graphe de modèles, où chaque nœud représente un simulateur (modèle couplé) et chaque arc symbolise le lien de dépendance qui unit deux simulateurs. La figure 1.9, montre un exemple de graphe de modèles, où chaque cercle représente un modèle atomique et chaque flèche le flux d'information qui transite entre eux.

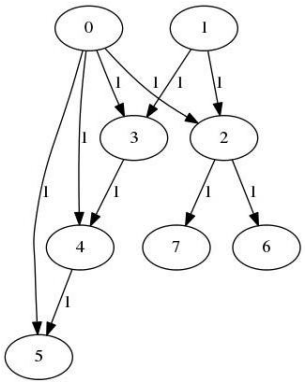


FIGURE 1.9 – Exemple de graphe de modèles, obtenu grâce à la mise à plat de la hiérarchie DEVS

Cette représentation d'une simulation DEVS fait appel à la notion mathématique de graphe. Celle-ci offre un ensemble de propriétés et de méthodes permettant de manipuler efficacement une telle structure. L'un des objectifs principaux de cette thèse étant de manipuler efficacement ce type d'objet dans le but d'optimiser la simulation en temps, il est important de bien présenter en quoi consiste la notion de graphe et quels sont les outils qu'elle offre.

1.3 Le partitionnement de graphes

La simulation distribuée DEVS consiste à répartir l'exécution des modèles atomiques parmi un ensemble de noeuds de calcul disponibles. Cette répartition se fait à partir du graphe de modèles issu de la mise à plat de la hiérarchie, en cherchant à répondre à cette simple question *Comment découper efficacement ce graphe pour garantir une distribution optimale ?* La réponse à cette question se trouve dans le partitionnement de graphes. En effet, l'application d'une méthode de partitionnement sur le graphe de modèles permet une subdivision de celui-ci et, en quelque sorte, une subdivision de la simulation. Les sous-graphes ainsi obtenus sont distribués sur les noeuds

de calcul et garantissent une exécution simultanée des modèles qui les composent. L'optimalité de la distribution repose en grande partie sur la qualité de ce partitionnement. D'où l'importance de bien cibler le problème, afin de fournir une méthode de partitionnement répondant au mieux à celui-ci.

Le partitionnement de graphe n'est pas chose nouvelle dans le domaine des mathématiques. Ce problème N-P complet a fait l'objet de nombreuses recherches et a donné naissance à de nombreuses méthodes dans le but d'approcher la meilleure solution. Ces problèmes peuvent se classer en deux grandes catégories : celle où l'on cherche à avoir des parties de même poids, on parlera de partitionnement ***contraint*** et celle où au contraire le poids des parties n'a pas d'importance, on parlera de partitionnement ***non contraint***. Il existe des méthodes proposant une réponse à chacun des ces problèmes, elles ont toutes en commun l'utilisation des notions issues de la théorie des graphes. La première phase de cet état de l'art a pour objectif de présenter en quoi consiste la théorie des graphes.

1.3.1 Formalisation mathématique de la théorie des graphes

La théorie des graphes est régie par un ensemble de concepts et de définitions formelles. Cette section fait état de l'ensemble des notions nécessaires à la compréhension des travaux menés dans cette thèse et les définit formellement.

Notion de graphe

Cette sous-section présente tous les principaux outils mathématiques liés aux graphes et à leur définition. La première et plus fondamentale notion de la théorie des graphes concerne bien évidemment la définition du graphe en lui-même :

Définition 9 (Graphe) : Soit V un ensemble sommets, nommé en référence à son nom en anglais *vertex*, qui se compose $n_V \in N^{*+}$ éléments et E un ensemble d'arêtes, nommé en référence à son nom anglais *edge*, qui se compose de $n_E \in N^{+}$ couples d'éléments de V. On appelle graphe G le couple (V,E).

Soient deux sommets $v_1, v_2 \in V$, tels que v_1 soit connecté à v_2 dans ce sens, et uniquement dans ce sens, un graphe contenant de telles règles est dit *orienté*. Au contraire, si un graphe ne contient aucune règle de connexion, en terme de sens, celui-ci est dit *non orienté*. On peut aussi distinguer, entre autre, une sous catégorie au graphe orienté : le graphe *bidirectionnel*. Celui-ci existe à condition que pour tout couple $v_1, v_2 \in V$, v_1 soient connectés à v_2 et réciproquement. Voici une définition formelle des ces graphes :

Définition 10 (Graphe orienté, non orienté) : Soit un graphe $G = (V,E)$. Si $\forall (x,y) \in E$, $(y,x) \in E$, alors le graphe est dit non orienté et les éléments de E sont appelés arêtes du graphe. Dans ce cas, on note indifféremment une arête : $e \in E$, $(v,v') \in E$ avec v et v' dans V, ou encore (V',V). Dans le cas contraire, le graphe est dit orienté et les éléments de E sont appelés arcs du graphe.

Il est possible d'associer une pondération aux arcs et aux nœuds dans le but de pénaliser ou d'accentuer leur importance dans le graphe. Cette notion est fondamentale pour le partitionnement de graphe. C'est un complément d'information indispensable pour qu'un graphe puisse refléter au mieux le processus qu'il représente. Les graphes incluant cette pondération portent le nom de graphes valués ou pondérés.

Définition 11 (Graphe valué ou pondéré) : Soit un graphe $G = (V,E)$. On dit que le graphe est valué (ou pondéré) si à chaque élément e de E est associée une valeur entière $poids(e) \in N^*$. La valeur $poids(e)$ est appelée le poids de e. Par extension, on considère qu'un couple de sommets $(v,v') \in V^2$ tel que $(v,v') \notin E$ possède un poids nul : $poids(v,v') = 0$. Le poids d'un sous-ensemble X d'éléments de E, est la somme des poids des éléments de X :

$$poids(X) = \sum_{e \in X} poids(e)$$

De même, on associe parfois aux éléments v de V un entier strictement positif appelé le poids de V et noté $poids(v)$.

Il est également possible d'associer à chaque sommet du graphe un nom, appelé index, permettant ainsi de se repérer facilement dans le graphe. Cet index peut être une valeur numérique, une chaîne de caractères ou même une simple lettre.

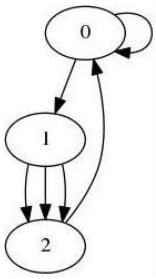


FIGURE 1.10 – Exemple de graphe avec boucle et arcs multiples

Il existe différentes formes et particularités pour un graphe, certains peuvent posséder des sommets dont l'arête se rattache à lui-même, ou encore deux sommets reliés par des arêtes multiples. La figure 1.10 illustre ces notions.

Définition 12 (Boucle et arête multiple) : Une arête est appelée une boucle si ses deux extrémités sont identiques. Si deux arêtes possèdent les mêmes extrémités, alors on dit que l'arête est multiple et que ces deux arêtes sont parallèles. Dans ce cas, la multiplicité d'une arête est le nombre total de ses arêtes parallèles, y compris elle-même. Il en va de même pour les graphes orientés.

Certains graphes ont des structures particulières et sont soumis à certaines règles. C'est le cas des deux types de graphes présentés ci-dessous :

Définition 13 (Graphe simple) : Un graphe est dit simple s'il n'a ni boucle ni arête multiple.

Définition 14 (Graphe connexe) : Soit un graphe $G = (V, E)$. On dit que ce graphe est connexe si, quels que soient les sommets v et v' de V, il existe un chemin de v vers v'.

Par extension, un graphe non connexe est un graphe qui peut se décomposer en n sous-graphes connexes. C'est à dire, qu'il existe un ou plusieurs groupes de sommets isolés. La figure 1.11 donne un exemple de graphe non connexe car il est impossible de rejoindre le sommet 6 en partant du sommet 0. Cependant, les deux sous-graphes qui en résultent sont connexes. La notion de connexité est très importante pour la suite de nos travaux sur le partitionnement.

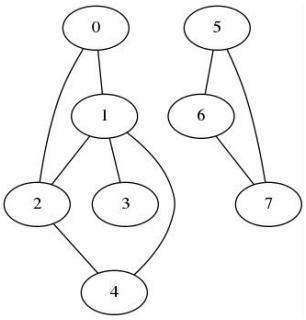


FIGURE 1.11 – Illustration de la connexité d'un graphe

Il existe un grand nombre de notions qui gravitent autour du concept de graphe. La sous-section suivante présente ces notions indispensables à la bonne compréhension des travaux menés dans cette thèse.

Notion annexe au graphe

Pour connaître la connexité d'un graphe, il est nécessaire de savoir si il existe des groupes de sommets qui ne sont pas rattachés au reste du graphe. Pour cela, il suffit de vérifier que tous les sommets du graphe sont connectés par une suite d'arêtes. Si une telle suite existe, on l'appellera chemin d'un graphe.

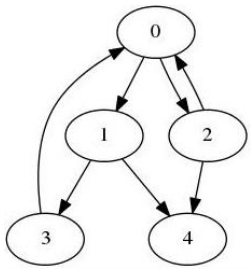
Définition 15 (Chemin d'un graphe) : Soient un graphe $G = (V, E)$, et deux sommets distincts $(v, v') \in V^2$. S'il existe une suite d'arêtes (ou d'arcs correctement orientés pour un graphe orienté) permettant d'atteindre v' à partir de v, alors on dit qu'il existe un chemin de v vers v' dans G.

De par sa forme, il est possible d'accéder et de connaître l'ensemble des voisins d'un sommet d'un graphe grâce aux arêtes qui le compose. Cette connaissance repose exclusivement sur la notion d'adjacence.

Définition 16 (Adjacence) : Soient un graphe $G = (V, E)$ et une arête $e = (v, v') \in E$. On dit que les sommets v et v' sont les sommets adjacents à l'arête e. De même, e est l'arête adjacente aux sommets v et v'.

Cette notion d'adjacence donne un cadre formel pour définir les interactions qui existent dans un graphe. Ces interactions sont regroupées dans une liste d'adjacence. Celle-ci permet de connaître, pour chaque sommet $v \in V$, la liste des sommets avec lesquels v possède une connexion.

La liste d'adjacence diffère en fonction de l'orientation de graphe. Le graphe ci-dessous, ainsi que les listes qui lui sont associées, donne un exemple de liste d'adjacence dans un cas orienté et non orienté.

Graphe Orienté	**Liste d'adjacence Orienté**	**Liste d'adjacence Non Orienté**
	$0 \rightarrow 1, 2$ $1 \rightarrow 3, 4$ $2 \rightarrow 0, 4$ $3 \rightarrow 0$ $4 \rightarrow$	$0 - 1, 2, 3$ $1 - 0, 3, 4$ $2 - 0, 4$ $3 - 0, 1$ $4 - 1, 2$

La liste d'adjacence ne comptabilise pas les arêtes multiples et les sommets voisins sont toujours triés par ordre croissant de leur index (où ordre alphabétique si leur index est une lettre). Le poids et l'index ne sont pas les seules informations que peut contenir un sommet, il existe aussi le degré :

Définition 17 (Degré d'un sommet) : Dans un graphe non orienté $G = (V, E)$, le degré d'un sommet $v \in V$ est le nombre d'arêtes auxquelles ce sommet appartient :

$$deg(v) = card(\{(v, v') \in E, v' \in V\}$$

Dans le cadre de graphes valués, le degré d'un sommet correspond à la somme des poids des arêtes auxquelles ce sommet appartient :

$$deg(v) = \sum_{(v,v') \in E} poids(v, v')$$

Pour ce qui est des graphes orientés, le degré d'un sommet correspond à la somme des poids des arcs entrants et sortants du sommet (ou au nombre d'arcs entrants et sortants).

Cette section fournit l'ensemble des notions théoriques nécessaires à la compréhension du problème du partitionnement de graphe. Les informations qui vont vous être présentées à partir de maintenant sont orientées autour du concept de *partition*.

Notion de partition

Le partitionnement de graphe est l'art de réaliser une partition de G, en respectant un ensemble de critères et de contraintes. Une partition peut être perçue comme le résultat du découpage d'un graphe G. Cependant, celle-ci ne correspond pas à un ensemble de sous-graphes, mais plutôt à un sous-ensemble de sommets ou d'arêtes. En terme général, on entend par partition la réunion des sous-ensembles des sommets du graphes, mais il est tout à fait possible de créer une partition

des arêtes du graphe comme le propose [Hol81]. Dans le cadre de cette thèse, nous considérons la partition au sens classique du terme :

Définition 18 (Partition) : Soit un ensemble V quelconque. Un ensemble P de sous-ensembles de V est appelé une partition de V si :

- Aucun élément de P n'est vide
- L'union des éléments de P est égal à V
- Les éléments de P sont deux à deux disjoints

Les éléments de P sont appelés les parties de la partition P.

Mais avant de présenter plus en détail en quoi consiste exactement le partitionnement de graphe et ses contraintes, il est nécessaire de savoir et de comprendre que le problème de partitionnement de graphe est un problème d'optimisation combinatoire.

1.3.2 L'optimisation sous contraintes

L'optimisation combinatoire consiste à déterminer la solution qui respecte au mieux une contrainte, généralement représentée par une fonction. Le nombre de solutions admissibles à un problème est généralement dénombrable, ce qui en fait un problème discret. L'optimisation combinatoire étant une branche des mathématiques discrètes, elle est un candidat naturel pour répondre à ce type de problèmes. Nous pouvons maintenant donner une définition formelle de ce qu'est un problème d'optimisation combinatoire.

Définition 19 (Problème d'optimisation combinatoire) : Un problème d'optimisation combinatoire se définit à partir d'un triplet (E_s, p, f) tel que :

- E_s est un ensemble discret appelé espace des solutions ou espace de recherche
- p est un prédicat de E_s, c'est à dire une fonction de E_s dans vrai, f aux
- $f : E_s \rightarrow R$ associe à tout élément $x \in E_s$ un coût $f(x)$. f est appelée fonction objectif ou fonction de coût.

Le rôle de p est de créer un ensemble $E_a = \{x \in E_s \text{ tel que } p(x) \text{ est vrai}\}$. L'ensemble E_a est appelé ensemble des solutions admissibles du problème. L'optimisation combinatoire consiste à trouver l'élément $\tilde{x} \in E_a$ qui minimise ou maximise f en fonction du problème :

$$f(\tilde{x}) = \min_{x \in E_a} f(x) \quad \text{ou} \quad f(\tilde{x}) = \max_{x \in E_a} f(x)$$

Chaque problème possède un spectre de solutions diverses et variées, qui peut se représenter par une ou plusieurs courbes en fonction du nombre de contraintes à respecter. Il est souvent difficile d'obtenir la meilleure solution, car pour un même problème, il existe de nombreuses méthodes de résolutions dont la convergence peut varier. Cette "meilleure solution" porte le nom ***d'optimum global***, qui correspond en réalité à l'extremum de la fonction que l'on cherche à optimiser. Chaque problème offre un ensemble de solutions dites satisfaisantes généralement proches de la meilleure

solution, elles portent le nom ***d'optimum local***. La définition qui suit, donne une définition formelle de la notion d'optimum.

Définition 20 (Optimum global, optimum local) : Soit un problème d'optimisation combinatoire (E_s, p, f) et E_a l'ensemble des solutions admissibles du problème induit par p. Soit $\tilde{x} \in E_a$

- si on peut prouver que $\forall x \in E_a$, $f(\tilde{x}) \leq f(x)$ ou $f(\tilde{x}) \geq f(x)$, alors on dira que $\tilde{x}$ est l'optimum global du problème, respectivement minimum et maximum global

- s'il existe un ensemble $F \subset E_a$, contenant $\tilde{x}$, et au moins deux éléments, tel que $\forall x \in F$, $f(\tilde{x}) \leq f(x)$ ou $f(\tilde{x}) \geq f(x)$, alors on dira que $\tilde{x}$ est un optimum local du problème, respectivement minimum et maximum local

La figure 1.12 montre un exemple de spectre de solution pour un problème mono-contraint. Cette courbe porte le nom de paysage. Grâce à celle-ci, il est possible de distinguer simplement les extremums locaux de l'extremum global.

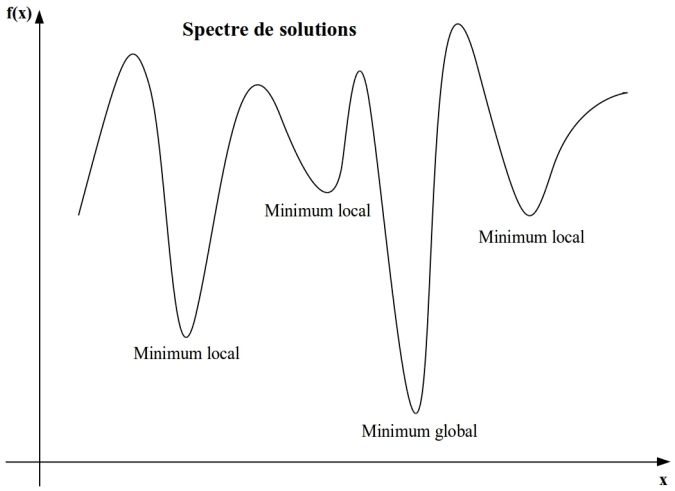


FIGURE 1.12 – Exemple de paysage d'une fonction de coût

Il n'est pas rare, au cours de la résolution d'un problème, qu'un algorithme d'optimisation se retrouve bloqué au niveau d'un minimum local. C'est pour cela, qu'il est très difficile de parvenir à l'optimum global. Pour parer ce problème, il est possible de perturber la solution obtenue dans le but de sortir de cet extremum local, et permettre à l'algorithme de reprendre la recherche de l'optimum global. Cette problématique constitue la base de la résolution sous contrainte qui est omniprésente pour le problème du partitionnement de graphes.

1.3.3 Le partitionnement de graphes

Comme nous l'avons vu précédemment dans 1.3.1, il existe deux écoles concernant le partitionnement d'un graphe. Notre choix s'oriente sur le partitionnement des sommets du graphe, dont voici la définition :

Définition 21 (Partition des sommets d'un graphe) : Soient un graphe $G = (V,E)$ et un ensemble de k sous-ensembles de V, noté $P_k = \{V_1,...,V_k\}$. On dit que P_k est une partition de G si :

- aucun sous-ensemble de S qui est élément de P_k n'est vide :

$$\forall i \in \{1,...,k\}, V_i = \emptyset$$

- les sous-ensembles de V qui sont éléments de P_k sont disjoints deux à deux :

$$\forall (i,j) \in \{1,...,k\}^2, i = j, V_i \cap V_j = \emptyset$$

- l'union de tous les éléments de P_k est V :

$$\bigcup_{i=1}^{k} V_i = V$$

Les éléments V_i de P_k sont appelés les parties de la partition. Le nombre k est appelé le nombre de parties de la partition.

Le partitionnement, tel qu'il est formulé ci-dessus, ne prend en compte ni la taille, ni le poids des parties générées. Or, cette information peut s'avérer primordiale en fonction de la nature du partitionnement que l'on cherche à réaliser. C'est notamment le cas pour le partitionnement contraint, dont l'objectif principal est de générer une partition de sommets du graphe équilibrée. Pour les graphes non valués, l'équilibre d'une partition se traduit par des sous-ensembles de sommets de même taille. Et pour les graphes valués, l'équilibre est respecté à condition que la somme des poids des sommets de chaque partie soit suffisamment proche. La balance d'une partition est un indicateur permettant d'évaluer l'équilibre d'une partition.

Définition 22 (Balance de partitionnement) : Soient un graphe $G = (V,E)$ et une partition $P_k = \{V_1, \cdots, V_k\}$ de ce graphe en k parties. Le poids moyen d'une partie V_i de P_k est

$$poids_{moy} = \left\lceil \frac{poids(V)}{k} \right\rceil$$

où $\lceil x \rceil \in R$ désigne le premier entier supérieur ou égal à x.
La balance $bal(P_k)$ de la partition P_k est égale à la division du poids de la partie de poids maximal de P_k par le poids moyen d'une partie :

$$bal(P_k) = \frac{\max_i poids(V_i)}{poids_{moy}}$$

Dans la littérature [ST97], une partition P_k, de balance $bal(P_k) \in [1, +\infty[$, est considérée comme équilibrée si $bal(P_k) \leq 1.05$. Il est évident que plus la balance est proche de 1, meilleur sera l'équilibre de celle-ci.

Le partitionnement de graphe est un problème d'optimisation combinatoire soumis à différentes fonctions objectifs. Le choix de ces fonctions dépendent totalement du problème que l'on

cherche à résoudre. La littérature propose trois grandes fonctions objectifs qui vous sont présentées dans la sous-section qui suit.

Les fonctions objectifs

Les fonctions objectifs pour le partitionnement de graphe s'articulent toutes autour de deux concepts : le coût de coupe entre les parties de la partition et le poids de ces parties. Soit un graphe $G = (V,E)$ et deux sous-ensembles $V_1, V_2 \subseteq V$, le coût de coupe entre ces deux sous-ensembles correspond à la somme des poids des arcs reliant les deux parties formées par ces sous-ensembles :

$$coupe(V_1,V_2) = \sum_{v_1 \in V_1, v_2 \in V_2} poids(v_1,v_2)$$

Soit $P_k = \{V_1, V_2, ..., V_k\}$ une partition de V en k parties. La première et la plus utilisée des fonctions objectifs pour le partitionnement de graphe est le *coût de coupe* introduite par Brian Kernighan et Shen Lin dans [KL70]. Cette fonction cherche à minimiser la somme des poids des arêtes entre les parties de la partition P_k :

$$\begin{aligned} coupe(P_k) &= \sum_{i<j} coupe(V_i,V_j) \\ &= \frac{1}{2} \sum_{i=1}^{k} coupe(V_i, V - V_i) \end{aligned}$$

Cette fonction objectif ne prend absolument pas en compte la taille et le poids des parties, et ne cherche en aucun cas à respecter la balance de partitionnement.

Yen-Chuen Wei et Chung-Kuan Cheng introduisent le *ratio de coupe* dans [WC89]. Cette fonction cherche à minimiser pour chaque partie le rapport entre son coût de coupe et son poids :

$$ratio(P_k) = \sum_{i=1}^{k} \frac{coupe(V_i, V - V_i)}{poids(V_i)}$$

Contrairement au coût de coupe, cette fonction objectif a pour objectif principal de minimiser la somme des poids des arêtes communicantes entre les parties de la partition et un second qui est de minimiser le déséquilibre entre les parties. Attention, même si cette fonction prend en compte l'équilibre des parties, ce n'est pas son objectif premier, il est donc rare d'obtenir une partition avec des parties équilibrées en utilisant cette fonction objectif.

La troisième et dernière fonction permet d'isoler les régions du graphe dont les sommets sont très liés. Ses auteurs, Jianbo Chi et Jitendra Malik, l'ont introduite dans [SJ00] et lui ont donné le nom de *coût de coupe normalisé*. Cette fonction cherche à minimiser, pour chaque partie, le rapport entre son coût de coupe et la somme du poids des arêtes adjacentes à au moins un de ses sommets.

$$norm(P_k) = \sum_{i=1}^{k} \frac{coupe(V_i, V - V_i)}{coupe(V_i, V)}$$
$$= \sum_{i=1}^{k} 1 - \frac{coupe(V_i, V_i)}{coupe(V_i, V)}$$

De par sa définition, cette fonction objectif ne peut, ou seulement dans de rare cas, fournir une partition équilibrée. Elle n'est donc pas recommandée pour un problème de partitionnement contraint. Le choix de la fonction objectif est fortement lié à la nature du problème de partitionnement.

1.3.4 Nature du partitionnement

Le partitionnement de graphes est utilisé dans de nombreux domaines tels que les mathématiques hautes performances (problèmes d'ingénierie, de calcul haute performance, de résolution de systèmes linéaires, etc ...), la conception de circuits intégrés [WC89], la segmentation d'images [FH04], la classification de textes [PF98] ou encore la répartition de charge pour les machines parallèles et distribuées [HQR15a] qui est en partie l'objet de cette thèse. Cette liste non exhaustive d'utilisation du partitionnement de graphes est soumis à certaines contraintes totalement dépendantes du problème à résoudre. Il est donc nécessaire d'adapter la méthode utilisée en fonction de la nature du problème de partitionnement rencontré. Comme nous l'avons vu plus haut (voir section 1.3), les problèmes de partitionnement peuvent se classer en deux grandes familles : le partitionnement contraint et le partitionnement non contraint. Cette sous-section présente chacune des ces natures et en explique les différences.

Le partitionnement contraint

Le problème du partitionnement de graphes contraint consiste à trouver une partition en k parties qui minimise une fonction objectif f tout en respectant la balance de partitionnement, c'est à dire de telle sorte que celle-ci soit unitaire. La balance est unitaire à condition que les parties aient le même poids, à une unité près. Or, si une partition P_k respecte parfaitement cette contrainte, alors son ratio de coupe devient égal à son coût de coupe, à un facteur multiplicatif près. En effet, dans ce cas :

$$ratio(P_k) = \sum_{i=1}^{k} \frac{coupe(V_i, V - V_i)}{poids(V_i)}$$
$$= \frac{k}{poids(V)} \sum_{i=1}^{k} coupe(V_i, V - V_i)$$
$$= \frac{k}{poids(V)} coupe(P_k)$$

Cette démonstration met en avant le fait qu'il est équivalent de chercher à minimiser le ratio de coupe et le coût de coupe pour ce type de problème. De plus, dans la section 1.3.3, il a été mis en

avant que l'utilisation du coût de coupe normalisé est en contradiction avec la nature contrainte du problème. En effet, il est quasiment impossible de chercher à respecter l'équilibre de la partition tout en cherchant à isoler les régions du graphe dont les sommets sont très liés. Ces remarques, font du coût de coupe, l'unique candidat pour la résolution du problème contraint dont voici la définition

Définition 23 (Partitionnement contraint) : Soient un graphe $G = (V, E)$, un nombre de parties k et une balance de partitionnement maximale, $balmax$. Soit l'espace des solutions E_s, défini par le problème général du partitionnement de graphes. L'ensemble E_a des solutions admissibles du problème est défini par :

$$E_a = \{P_i \in E_s \text{ tel que } card(P_i) = k \text{ et } bal(P_i) \leq balmax\}$$

Le problème du partitionnement contraint consiste à trouver $\tilde{x} \in E_a$ qui minimise la fonction de coût de coupe :

$$coupe(\tilde{x}) = \min_{x \in E_a} coupe(x)$$

Le partitionnement contraint possède une contrainte forte : l'équilibre de la partition et une contrainte faible : la minimisation du poids des arcs communiquants. Dans ce type de problème, la contrainte forte est toujours privilégiée au détriment de la contrainte faible.

Le partitionnement non contraint

Contrairement au problème contraint, le problème non contraint n'est pas soumis au respect de la balance de la partition. Son unique objectif est de minimiser l'une des fonctions objectifs citées dans 1.3.3, ou une fonction dérivée de celles-ci. Cependant, même si l'objectif principal n'est pas d'équilibrer la partition, certaines fonctions objectif intègrent intrinsèquement la notion d'équilibre afin d'éviter une dérive complète de la balance. C'est notamment le cas du ratio de coupe. Ainsi, contrairement au cas contraint, la plupart des problèmes omettent la contrainte sur la balance de partitionnement et n'énoncent que la fonction objectif à minimiser. La contrainte sur la balance de la partition n'est donc pas incluse dans la définition du partitionnement non contraint.

Définition 24 (Partitionnement non contraint) : Soient un graphe $G = (V, E)$ et un nombre de parties k. Soit l'espace des solutions E_s, défini par le problème général du partitionnement de graphes. L'ensemble E_a des solutions admissibles du problème est défini par :

$$E_a = \{P_i \in E_s \text{ tel que } card(P_i) = k\}$$

Le problème du partitionnement contraint consiste à trouver $\tilde{x} \in E_a$ qui minimise une fonction de coût f comme le coût de coupe normalisé ou le ratio de coupe :

$$f(\tilde{x}) = \min_{x \in E_a} f(x)$$

Ce problème étant soumis à moins de contraintes, l'ensemble des solutions admissibles E_a est plus grand que pour un partitionnement de nature contraint. Ce qui a pour effet de rendre plus

facile la découverte d'une partition qui satisfasse le problème de nature non contraint. Cependant, cela ne le rend pas plus simple à explorer pour autant. Il est important de bien déterminer la nature du problème à résoudre avant d'en commencer la résolution, car en fonction du choix, certaines solutions pourraient ne pas être accessibles.

Différence entre partitionnement contraint et non contraint

Ces deux types de problèmes de partitionnement ont une formulation assez proche mais les algorithmes utilisés pour les résoudre sont, eux, en général bien différents. Cette différence se trouve principalement dans leur implémentation et dans leurs objectifs. En effet, ces deux problèmes diffèrent principalement dans leur priorité :

- le partitionnement contraint privilégie le respect de la balance de partitionnement au détriment de la minimisation du coût de coupe. Une partition ayant un coût de coupe minimum ne serait pas retenue si elle ne respecte pas la contrainte d'équilibre.

- le partitionnement non contraint a pour but de minimiser le coût de coupe dans la limite de parties dont la taille est inversement proportionnelle à ce coût de coupe. Le principe étant d'éviter un déséquilibre trop grand des parties, sauf dans le cas où un coût de coupe est vraiment extrêmement faible. Ainsi, le but premier du cas non contraint est de minimiser le coût de coupe, et le second est d'éviter au mieux le déséquilibre la partition sans pour autant respecter la balance.

L'utilisation d'un algorithme pour résoudre un problème non contraint, sur un problème contraint donne une partition de mauvaise qualité car celle-ci n'est pas conçue pour respecter la balance de partitionnement. La remarque est également valable pour le cas inverse. C'est ce qu'a pu constater Charles Edmond Bichot dans son article [Bic07].

1.4 Les grandes catégories de méthodes de partitionnement

La résolution des problèmes de partitionnement s'oriente autour de plusieurs grandes catégories de méthodes. Celles-ci dépendent de la nature du problème à résoudre mais aussi du niveau de finesse recherché et des paramètres du graphe (taille, connexité, etc ...). Avant de fixer un choix sur une ou plusieurs catégories de méthode, il a été nécessaire de réaliser une étude sur les principales méthodes de partitionnement. Le rôle de cette section est de présenter ces méthodes.

1.4.1 Méthodes de classification

Les méthodes de classification cherchent à construire une partition d'un ensemble d'objets dont on connaît les distances deux à deux. La classification est une branche très importante de l'informatique car elle a de nombreuses applications. Les problèmes de classification sont très proches de ceux du partitionnement de graphe comme le montre [DGK04] et [DGK07]. Ce type de méthode permet de résoudre les problèmes de nature non contraint, en laissant libre choix de la fonction objectif à minimiser.

La classification repose sur plusieurs techniques dont les plus classiques sont :

- la méthode des centres mobiles [KMN+02]. C'est une méthode itérative qui consiste à calculer le centre de gravité de chaque partie de la partition, puis à déplacer les éléments en fonction de leur distance par rapport au centre de gravité de chaque partie. Ainsi, au cours de chaque itération, des sommets sont ajoutés ou déplacés vers la partie dont la distance au centre de gravité est la plus faible. Cette méthode est rapide mais nécessite la notion de distance.

- les méthodes hiérarchiques [GF05]. Ces méthodes créent des arbres dont chaque niveau constitue une partition. Elles reposent toutes sur le même principe : créer un ensemble de partitions réparties hiérarchiquement en classes possédant de moins en moins de parties. Chaque nouvelle partition est obtenue par regroupements successifs de parties du niveau précédant de la hiérarchie. Il existe des méthodes hiérarchiques ascendantes ou descendantes. Dans le cas des méthodes ascendantes, les deux individus les plus proches sont regroupés pour former un sommet de l'arbre et ainsi de suite, jusqu'à ce qu'il n'y ait plus qu'un seul individu.

Les méthodes de classification reposent sur une fonction de distance, ou une fonction de dissimilitude, entre tous les couples d'éléments de l'ensemble à classer. Cependant, il n'existe pas de relation de distance immédiate entre tous les éléments d'un graphe quelconque. Si une relation de distance peut être créée, elle est en général très coûteuse à mettre en place, particulièrement pour les graphes non connexes. Ainsi, les méthodes de classification pour le partitionnement de graphe sont en général inutilisables.

1.4.2 Les métaheuristiques

Les métaheuristiques sont des méthodes "gloutonnes", qui ont l'avantage d'être efficaces même sur des problèmes très complexes mais qui nécessitent une paramétrisation longue pour obtenir des résultats performants. Ces méthodes sont généralement coûteuses en temps de calcul. Elles ont fait leur apparition dans les années 1980 pour résoudre des problèmes d'optimisation difficile, en s'inspirant de processus naturels.

Définition 25 (Métaheuristique) : Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficiles pour lesquels on ne connaît pas de méthode classique plus efficace. Le principe de base d'une métaheuristique est de parcourir l'espace des solutions à la recherche de son minimum global en utilisant des mécanismes lui permettant de s'extraire des minimas locaux. Elles s'inspirent souvent de phénomènes physiques.

Une métaheuristique n'est utilisée que lorsque aucune autre méthode n'est plus efficace qu'elle. Car si une telle méthode existe, elle sera obligatoirement plus rapide qu'une métaheuristique. Le mode de fonctionnement de celle-ci est de perturber la solution obtenue afin de voir s'il en existe une meilleure. L'objectif de cette perturbation est de sortir d'une solution locale dans le but de trouver la solution globale du problème. Le coût en temps de ces méthodes provient de cette recherche perpétuelle de la solution globale. Des critères sont mis en place afin de mettre fin à cette

recherche itérative, comme par exemple un critère d'arrêt qui dépend du nombre d'itérations sans trouver de meilleure solution. Ces critères sont liés au type de métaheuristique utilisé.

Les métaheuristiques peuvent être classées en deux grandes familles, celles de voisinage et celles à base de population. Les métaheuristiques de voisinage ont en commun d'avoir un mécanisme d'acceptation de solutions dégradées leur permettant de s'extraire des minima locaux. Et celles à base de population disposent d'un mécanisme collectif pour se sortir des minima locaux. Voici une présentation des deux principales métaheuristiques :

- le recuit simulé [KGV83] est une métaheuristique de voisinage qui s'inspire de la technique expérimentale du recuit utilisée en métallurgie. Celle-ci permet d'obtenir un état d'énergie minimale du métal. Son principe est simple : partant d'un état initial quelconque, un nouvel état est créé à partir de l'état précédent par une modification élémentaire de cet état. Il est accepté si son énergie est plus faible, sinon il est accepté avec une certaine probabilité. Ce schéma est réitéré en remplaçant l'état initial par l'état accepté tant qu'un "équilibre thermodynamique" n'est pas atteint. Lorsque cet équilibre est atteint, la température est diminuée. Le processus est renouvelé tant que le système n'est pas figé. Dans le cadre du partitionnement de graphe, l'état correspond à une partition de l'ensemble des sommets du graphe et l'énergie à la fonction objectif à minimiser. La conservation des solutions de qualité moindre empêche l'algorithme d'être bloqué au niveau d'un extremum local. Il est possible de comparer cette méthode à un algorithme d'affinage de partition, car son objectif consiste à trouver une partition optimale en partant d'une partition initiale. Or, plus la partition initiale est de bonne qualité, plus vite l'algorithme converge vers l'extremum global. L'algorithme du recuit simulé est présenté et décrit en détail dans de nombreux ouvrages tels que [SD89] [DPST03].

- la colonie de fourmis [DCG99] est une métaheuristique à base de population qui s'inspire du comportement des fourmis lorsqu'elles sont à la recherche de nourriture. En se déplaçant une fourmi dépose des phéromones, une substance olfactive et volatile, sur son chemin. Les fourmis se dirigent de manière probabiliste en tenant compte de la quantité de phéromone se trouvant autour d'elles et de la quantité qui a été précédemment déposée par les autres membres de la colonie. Plus la quantité de phéromone sur un chemin est grande, plus une fourmi a tendance à suivre ce chemin. Cependant, comme la phéromone s'évapore progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue continuellement. Les algorithmes de colonies de fourmis s'inspirent de la capacité collective qu'ont les fourmis de résoudre certains problèmes, qu'elles ne pourraient pas résoudre individuellement étant donné leurs capacités limitées. Pour le partitionnement de graphes, cette méthode consiste à générer k colonies de fourmis (une pour chaque partie de la partition). En partant d'une partition initiale, où chaque partie est une colonie, on applique le principe des phéromones pour faire évoluer les colonies. Lorsque les colonies sont stabilisées, on attribue à chaque colonie une partie de la partition.

Ces méthodes sont facilement adaptables au partitionnement de graphes [BS13]. Il existe d'autres métaheuristiques adaptables au partitionnement de graphes, comme par exemple : la recherche tabou, les méthodes évolutionnaires et les méthodes hybrides. Ces méthodes ont toutes en commun

de nécessiter une partition initiale pour pouvoir être appliquées. Elles ont l'avantage de pouvoir résoudre les deux types de problèmes de partitionnement au prix de quelques modifications dans leur implémentation. Il est préférable d'utiliser ces méthodes lorsqu'on veut garantir la convergence vers la solution optimale au détriment des temps de calcul qui peuvent vite devenir très importants.

1.4.3 Méthodes spectrales

Les méthodes spectrales sont des méthodes de partitionnement reposant exclusivement sur l'algèbre linéaire des matrices. Elles ont la particularité d'offrir une unique solution à un problème par résolution de systèmes linéaires. Contrairement à la grande majorité des méthodes de partitionnement, les méthodes spectrales ne sont pas soumises à un quelconque tirage aléatoire pour déterminer cette solution. Cette rigueur peut, dans certains cas, contraindre le problème et exclure des solutions accessibles par des méthodes plus souples. Le cadre formel et la rigueur mathématique de cette catégorie de méthodes en font des candidats sérieux pour répondre au problème de partitionnement.

Outre la liste d'adjacence (voir section 1.3.1), il est possible de définir les relations de dépendance entre les sommets d'un graphe à l'aide de deux matrices : matrice d'adjacence et matrice des degrés. La matrice d'adjacence concatène le poids des arcs reliant chaque paire de sommets et la matrice des degrés est une matrice diagonale retournant le degré de chaque sommet du graphe. Ces matrices sont formellement définies comme suit :

Définition 26 (Matrice d'adjacence et des degrés) : Soit un graphe simple $G = (V,E)$.
La matrice M_{Adj} de dimension $n_V * n_V$, telle que $\forall (i,j) \in \{1,...,n_V\}^2$:

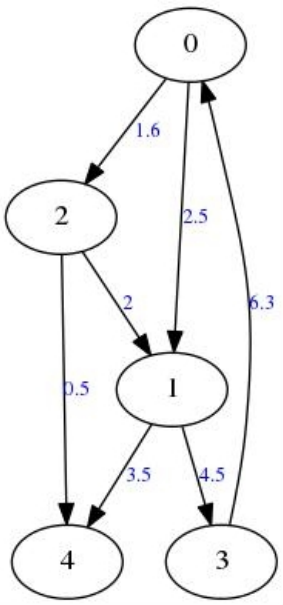
$$(M_{Adj})_{ij} = \begin{cases} 0 \text{ si } i = j \\ poids(v_i, v_j) \text{ sinon} \end{cases}$$

est appelée matrice d'adjacence du graphe G.
La matrice M_{Deg} de dimension $n_V * n_V$, telle que $\forall (i,j) \in \{1,...,n_V\}^2$:

$$(M_{Deg})_{ij} = \begin{cases} deg(i) = \sum_{k=1}^{n_S} poids(v_i, v_k) \text{ si } i = j \\ 0 \text{ sinon} \end{cases}$$

est appelée matrice des degrés du graphe G.

Cette structure a l'avantage de contenir plus d'information de la liste d'adjacence mais elle est donc plus lourde à manipuler. À titre d'exemple, reprenons la figure 1.3.1 en attribuant à chaque arcs un poids, dans le but d'illustrer la notion de matrice d'adjacence et matrice des degrés. Comme on peut le constater dans l'exemple ci-dessous, lorsqu'un un arc n'est pas présent dans le graphe, le poids qui lui est associé dans la matrice d'adjacence est nul. Il est simple de déduire la matrice des degrés à partir de la matrice d'adjacence, les valeurs prises par la diagonale de M_{deg} sont égales à la somme des valeurs de chaque ligne de M_{adj}.



Graphe orienté

Matrice d'adjacence

$$\begin{pmatrix} 0 & 2.5 & 1.6 & 0 & 0 \\ 0 & 0 & 0 & 4.5 & 3.5 \\ 0 & 2 & 0 & 0 & 0.5 \\ 6.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrice des degrés

$$\begin{pmatrix} 4.1 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0 & 6.3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Graphe non orienté

Matrice d'adjacence

$$\begin{pmatrix} 0 & 2.5 & 1.6 & 6.3 & 0 \\ 2.5 & 0 & 2 & 4.5 & 3.5 \\ 1.6 & 2 & 0 & 0 & 0.5 \\ 6.3 & 4.5 & 0 & 0 & 0 \\ 0 & 3.5 & 0.5 & 0 & 0 \end{pmatrix}$$

Matrice des degrés

$$\begin{pmatrix} 10.4 & 0 & 0 & 0 & 0 \\ 0 & 12.5 & 0 & 0 & 0 \\ 0 & 0 & 4.1 & 0 & 0 \\ 0 & 0 & 0 & 10.8 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}$$

Le type de graphe (orienté ou non) modifie totalement ces deux matrices. Pour les graphes non orientés, la matrice d'adjacence est symétrique, ce qui n'est pas le cas pour les graphes orientés. Dans le cas d'un graphe orienté, la matrice des degrés correspond en réalité à la somme des poids des arcs sortant d'un nœud, ce qui diffère légèrement avec la définition 17 donnée en section 1.3.1.

La théorie mathématique offre un outil puissant pour le problème du partitionnement de graphes. Celui-ci peut être ramené à la résolution d'un système numérique $Mx = \lambda x$ au prix de quelques approximations. Résoudre ce genre de système numérique consiste à trouver une base orthogonale de vecteurs propres de la matrice M. Ce qui implique que le problème du partitionnement de graphe peut être résolu en trouvant une famille de valeurs propres λ et une famille de vecteurs propres x d'une matrice. Cette matrice en question, est la matrice Laplacienne définie comme suit :

Définition 27 (Matrice laplacienne) : Soit un graphe $G = (V, E)$. La matrice

$$M_{Lap} = M_{Deg} - M_{Adj}$$

est appelée matrice Laplacienne (ou de Laplace) de G.

Tout d'abord, il est nécessaire de faire quelques rappels d'algèbre linéaire concernant les valeurs propres et les vecteurs propres afin de faciliter la compréhension de cette section.

Définition 28 (Vecteur propre, valeur propre) : Soit E_v un espace vectoriel sur un corps K. Soit u un endomorphisme de E_v. Un vecteur x de E_v tel qu'il existe λ dans K tel que $u(x) = \lambda x$ est appelé vecteur propre de u. Dans ce cas, λ est appelé valeur propre de u.

La méthode spectrale doit son nom au théorème spectral de l'algèbre linéaire. Son utilisation pour la résolution du problème du partitionnement de graphe n'est pas nouveau [DH72]. De nombreux articles présentent l'application de la méthode spectrale au partitionnement de graphe, tant

pour le problème du partitionnement contraint [HL95], [PSL90] que pour celui du partitionnement non contraint [HK92], [DGK04] .

Dans un premier temps, intéressons nous à la bissection d'un graphe $G = (V,E)$ en deux ensembles de sommets V_1 et V_2. Soit x un vecteur de taille n tel que :

$$\forall v_i \in V,\ x_i = \begin{cases} 1 & \text{si } v_i \in V_1 \\ -1 & \text{si } v_i \in V_2 \end{cases}$$

En reprenant le coût de coupe entre V_1 et V_2 défini en section 1.3.3 on a :

$$\begin{aligned} coupe(V_1,V_2) &= \sum_{v_1 \in V_1, v_2 \in V_2} poids(v_1,v_2) \\ &= \frac{1}{2} \sum_{(v_i,v_j) \in E} (x_i - x_j)^2 poids(v_i,v_j) \\ &= \frac{1}{2} x^T M_{Lap}\, x \end{aligned}$$

Minimiser le coût de coupe d'une bissection revient à trouver un vecteur x qui minimise $x^T M_{Lap}\, x$. Autrement dit, ce problème consiste à résoudre le système linéaire :

$$M_{Lap}\, x = \lambda x$$

Résoudre ce système numérique consiste à trouver la plus petite valeur propre λ de la matrice M_{Lap} et le vecteur propre qui lui est associé.

Méthodes de résolution

Le problème du partitionnement de graphe peut se résoudre par la recherche des vecteurs propres de la matrice Laplacienne. Plusieurs méthodes, présentées dans [GL96], peuvent être utilisées dans le but de trouver les vecteurs propres de la matrice Laplacienne :

- l'algorithme itératif de Lanczos
- la méthode itérative du quotient de Rayleigh
- l'algorithme de la décomposition QR
- l'algorithme itératif de Jacobi

Les valeurs propres $(\lambda)_i$ issues de la matrice M_{Lap} ont toutes en commun d'être positives ou nulles. La plus petite valeur propre est $\lambda_1 = 0$, à qui on associe le vecteur propre trivial $x = (1,\cdots,1)^T$. Il est possible de trier les valeurs propres par ordre croissant $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, ainsi que les vecteurs propres qui leur sont associés. Ces valeurs et vecteurs propres possèdent certaines propriétés qui sont notamment très utiles pour le partitionnement de graphes.

La première personne à en avoir étudié les propriétés est Miroslav Fiedler dans [Fie75]. C'est pour cela, que ces vecteurs propres portent le nom de vecteurs de Fiedler. Le second vecteur de Fiedler, qui est associé à la seconde plus petite valeur propre, permet d'obtenir une bissection de coût de coupe minimum d'un graphe $G = (V,E)$ en deux sous-ensembles V_1, V_2 tels que $V_1 \cup V_2 =$

V. Pour cela, il suffit de discrétiser les valeurs du vecteur vers $\{-1;1\}$ pour répartir les sommets du graphe dans les deux ensembles. Ce processus est généralisable au partitionnement de la forme 2^i. En effet, les $i+1$ premiers vecteurs de la famille de Fiedler de la matrice Laplacienne de G permettent de trouver une 2^i-partition de G. Pour cela, il suffit de discrétiser chaque vecteur de Fiedler vers $\{-1;1\}$, puis de créer les bissections correspondantes. Les modalités de discrétisation dépendent fortement de la nature du partitionnement.

Cas du partitionnement contraint

Le partitionnement contraint consiste à trouver une partition de G qui soit équilibrée et dont le coût de coupe soit minimal. Nous avons vu ci-dessus que la bissection s'obtient par discrétisation du second vecteur de Fiedlder et qu'il est possible de généraliser ce processus pour le partitionnement de la forme 2^i. Le problème est que seuls les trois premiers vecteurs de Fiedler (sans compter le vecteur trivial) apportent des résultats fiables.

Afin de respecter la balance de partitionnement, la discrétisation du vecteur de Fiedler x vers $\{-1;1\}$ se fait comme suit :

- trie des valeurs de x par ordre croissant
- discrétisation des $\frac{n}{2}$ premières valeurs vers -1
- discrétisation des $\frac{n}{2}$ dernières valeurs vers 1

Une telle discrétisation garantit à coup sûr le respect de la balance de partitionnement. Pour le 2^i-partitionnement, le même processus est appliqué à chaque vecteur de Fiedler, en respectant les modifications imposées par les bissections précédentes au niveau de la balance de partitionnement. C'est le respect de ces modifications qui limite l'utilisation des vecteurs de Fiedler. Car au-delà du troisième, il devient quasiment impossible de respecter les modifications. Il est cependant possible d'obtenir une partition $k = 2^i \geq 8$ en utilisant une approche par bissection récursive, ou par bissection itérative hiérarchique (voir 1.4.4).

Cas du partitionnement non contraint

Dans le cas du partitionnement non contraint, on cherche à obtenir une partition qui respecte l'une des deux fonctions objectifs suivantes : le ratio de coupe et coût de coupe normalisé. En fonction du choix de la fonction objectif, une modification est apportée au système à résoudre. Pour le ratio de coupe, le système à résoudre devient :

$$M_{Lap}x = \lambda M_{Adj}\, x$$

Et pour la coût de coupe normalisé :

$$M_{Lap}x = \lambda M_{Deg}\, x$$

Comme pour le cas contraint, le partitionnement s'obtient par discrétisation des vecteurs de Fiedler. Cependant, dans ce cas, il n'est pas nécessaire d'obtenir des parties de même poids. La discrétisation des vecteurs est donc totalement différente :

- discrétisation de toutes les valeurs $x_i < 0$ vers -1

- discrétisation de toutes les valeurs $x_i > 0$ vers 1

L'un des inconvénients des méthodes spectrales est qu'elles utilisent des algorithmes de recherche de vecteurs propres qui sont gourmands en temps de calcul et en espace mémoire. Afin de remédier à ce problème, il est conseillé de combiner la méthode spectrale à un algorithme multi-niveaux pour les graphes de plus de 10000 sommets. De plus, la méthode spectrale étant une méthode de recherche globale, il est conseillé d'appliquer un algorithme d'affinage à la partition obtenue afin d'en améliorer la qualité. La méthode multi-niveaux, ainsi que les méthodes d'affinages, sont présenter en section 1.5.3.

1.4.4 Méthodes d'expansion de région

Les méthodes d'expansion de région sont des méthodes déterministes reposant principalement sur la notion de voisinage. Elles ont toutes en commun d'être des méthodes de bissection de graphe conçues pour respecter au mieux la balance de partitionnement. Ceci fait de ces méthodes des candidats sérieux pour la résolution du problème de partitionnement contraint. Leur application au problème de k-partitionnement nécessite l'utilisation d'une méthode de bissection récursive. Cependant, nous avons préféré une approche itérative hiérarchique à une approche récursive. Le principe de fonctionnement de celle-ci est similaire à une approche récursive mais diffère dans son implémentation.

Algorithme de bissection itérative hiérarchique

Nombre de méthodes sont initialement conçues pour résoudre un problème de bissection de graphes. Or, dans la pratique, le partitionnement de graphes ne se limite pas à un 2-partitionnement. Pour être complète, une méthode se doit de répondre correctement à un problème de k-partitionnement, où k est généralement de la forme 2^i. Nous verrons par la suite qu'il est possible de généraliser le processus pour un k quelconque.

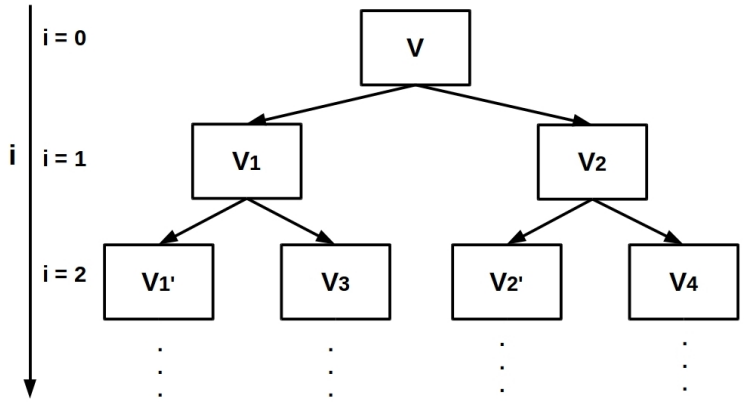


FIGURE 1.13 – Exemple de hiérarchie de partition obtenue pour un 4-partitionnement

L'algorithme de bissection itérative hiérarchique permet d'étendre les méthodes de bissection au k-partitionnement. Comme son nom l'indique, cette méthode repose sur une hiérarchie de parti-

tions obtenue par application itérative d'une méthode de bissection sur chacun des sous-ensembles de la partition précédemment obtenue. Dans certains cas, il est possible d'exprimer k sous la forme d'une puissance de 2, c'est à dire $k = 2^i$. La hiérarchie de partitions obtenue se compose de $i+1$ niveaux, où i s'obtient comme suit :

$$\begin{aligned} 2^i &= k & i\ln(2) &= \ln(k) \\ e^{i\ln(2)} &= k & i &= \frac{\ln(k)}{\ln(2)} \end{aligned}$$

Pour $k = 2^i$, le nombre de bissections nécessaires pour atteindre la partition souhaitée est de $\sum_{p=1}^{i} p$

Au niveau $i = 0$, la partition P se compose d'un unique ensemble contenant l'intégralité des sommets du graphe G, c'est à dire $P = \{V\}$. L'application d'une première bissection sur l'ensemble V donne naissance à une bi-partition $P = \{V_1, V_2\}$, qui correspond au niveau $i = 1$ de la hiérarchie. Le niveau $i+1$ s'obtient en appliquant une bissection sur chaque ensemble du niveau i. Ainsi le dernier niveau correspond au résultat du k-partitionnement souhaité. Dans cet exemple $k = 4 = 2^2$, ce qui implique une hiérarchie à 3 niveaux et à 3 bissections pour obtenir la 4-partition $P = \{V_1, V_2, V_3, V_4\}$.

Algorithme 6 Algorithme de bissection itérative hiérarchique pour le 2^i-partitionnement

PROCÉDURE: Bissection_Itérative_Hiérarchique_2i
ENTRÉES: $G = (V, E)$, k le nombre de parties
  Calcul de la profondeur $i_{max} = \frac{\ln(k)}{\ln(2)}$
  Initialisation de la partition $P = V_1$, où $V_1 = V$
  Pour i allant de 0 à i_{max} **faire**
    Pour j allant de 1 à 2^i **faire**
      Bissection de l'ensemble V_j
      Ajout de V_{j+i} à P
      Si Modification du graphe nécessaire **Alors**
        Mise à jour de G par suppression des arcs reliant les $V_k \subset P$
      Finsi
    Fin pour
  Fin pour
  Sorties: P

L'algorithme 6 présente le fonctionnement d'un algorithme de bissection itérative hiérarchique reposant sur le principe de hiérarchie de partitions illustré par la figure 1.13. Certains algorithmes de partitionnement nécessitent une modification du graphe après chaque bissection. Cette modification consiste, en général, à supprimer les arcs reliant les parties de la partition P. C'est notamment le cas pour les algorithmes d'expansion de région, qui reposent sur un mécanisme de propagation au sein du graphe. Or, partant d'un sommet $v_i \in V_i$, on ne doit pas pouvoir accéder aux sommets $v_j \in V_j$. Sans quoi, l'algorithme pourrait attribuer un même sommet à deux sous-ensembles de la partition, ce qui va à l'encontre de la définition de partition présentée en 1.3.1.

Il est possible d'adapter cet algorithme au k-partitionnement quelconque en ajoutant une étape supplémentaire. La bissection itérative hiérarchique pour le k-partitionnement consiste à décom-

poser le processus en un partitionnement de la forme 2^i, puis d'appliquer une bissection sur autant de sous-ensembles de la partition que nécessaire pour atteindre le k-partitionnement souhaité. La première étape consiste à déterminer le $2^i < k$ plus proche de k et $r = k - 2^i$, où r correspond au nombre d'ensemble restant à partitionner. Ainsi, l'algorithme 6 est appliqué jusqu'à atteindre le 2^i-partitionnement, puis r ensembles parmi les 2^i disponibles sont soumis à une nouvelle bissection. Ces ensembles sont choisis de façon à endommager le moins possible la balance de partitionnement. Il est naturel de dire qu'une telle approche endommage systématiquement la balance de partitionnement, même si celui-ci peut être minimisé par un choix judicieux. Cependant, il est possible de ré-équilibrer celle-ci à l'aide d'algorithme d'équilibrage de charge (voir la section 1.5.4) et d'affinage (voir la section 1.5.3). L'algorithme 7, présente le fonctionnement de la méthode de bissection itérative hiérarchique pour un k-partitionnement quelconque.

Algorithme 7 Algorithme de bissection itérative hiérarchique pour le k-partitionnement quelconque

PROCÉDURE: Bissection_Itérative_Hiérarchique_K
ENTRÉES: $G = (V, E)$, k le nombre de parties
  Calcul du $2^i < k$ plus proche de k
  Calcul du nombre de bissections restantes à réaliser $r = k - 2^i$
  Application de l'algorithme 6 pour obtenir la 2^i-partition
  Sélection des ensembles (V_p) de poids maximum
  Pour i allant de 1 à r **faire**
    Bissection de l'ensemble $V_i \subset (V_p)$
    Ajout de V_{i+1} à P
    Si Modification du graphe nécessaire **Alors**
      Mise à jour de G par suppression des arcs reliant les $V_k \subset P$
    Finsi
  Fin pour
  Sorties: P

En général, ce type d'approche a pour effet de réduire l'accès à certaines solutions et donc d'être légèrement moins performantes qu'une méthode directe. Cependant, il existe des procédés pour contourner ce problème.

La méthode Graph Growing Partitioning (GGP)

Le graph growing algorithm (GGP) est une méthode de bissection de graphe introduite par George Karypis et Vipin Kumar dans [KK98a]. Son principe est simple, il repose sur l'évolution de deux ensembles de sommets : V_{ori} contient l'intégralité des sommets du graphe G et V_{dest} initialement vide est conçu pour accueillir les sommets de V_{ori}. Ces ensembles sont construit pour respecter, à tout moment de l'exécution de l'algorithme, les conditions suivantes :

$$\begin{cases} V_{ori} \cup V_{dest} = V \\ V_{ori} \cap V_{dest} = \emptyset \end{cases}$$

L'algorithme débute par le tirage aléatoire d'un sommet v_i du graphe G. Celui-ci est déplacé de V_{ori} vers V_{dest}. L'algorithme repose sur un processus itératif simple, les sommets adjacents aux

sommets de l'ensemble V_{dest} sont déplacés de V_{ori} vers V_{dest}, à condition qu'ils n'y appartiennent déjà, jusqu'à ce que l'ensemble V_{dest} atteigne la moitié du poids total des sommets du graphe. L'algorithme fourni une partition $P = \{V_{ori}, V_{dest}\}$ théoriquement équilibré. Cependant, nous verrons par la suite que cette affirmation n'est pas toujours vraie.

Cette méthode est efficace sur des problèmes de faibles tailles (inférieur à 200 sommets), mais l'est beaucoup moins sur des problèmes de plus grandes taille.

La méthode Greedy Graph Growing Partitioning (GGGP)

La méthode GGGP est une amélioration de la méthode GGP, proposée par ses auteurs dans [KK98a]. L'amélioration réside dans la sélection des sommets adjacents à ceux de V_{dest}. Pour cela, un nouvel ensemble de sommet est ajouté : V_{adj} qui est l'ensemble des sommets adjacents à ceux présents dans V_{dest}. Cet ensemble doit respecter, à tout moment de l'exécution de l'algorithme, la condition suivante :

$$\forall v_i \in V_{adj}\ v_i \notin V_{dest}$$

L'algorithme débute par le tirage aléatoire d'un sommet v_i du graphe G. Celui-ci est déplacé de V_{ori} vers V_{dest}. L'ensemble V_{adj} est ensuite mis à jour en ajoutant tous les sommets adjacents à $v_i \notin V_{dest}$ et n'étant pas déjà présents dans V_{adj}. Contrairement à la GGP, seul le sommet qui minimise au mieux le coût de coupe est déplacé de V_{ori} vers V_{dest} et est supprimé de V_{adj}. Le processus est itéré jusqu'à ce que l'ensemble V_{dest} atteigne la moitié du poids total des sommets du graphe. L'algorithme 8 présente le fonctionnement de cette méthode.

Algorithme 8 Algorithme Greedy Graph Growing Partitioning (GGGP)

PROCÉDURE: GGGP
ENTRÉES: $G = (V, E)$
  Initialisation $V_{ori} = V$ contient tous les sommets de V
  Initialisation V_{dest} et V_{adj} sont vides
  Initialisation $poids_{dest} = 0$ du poids de l'ensemble destination à 0
  Calcul du poids moyen $poids_{moy} = \frac{poids(V)}{2}$
  Tirage aléatoire d'un sommet $v_i \in V_{ori}$
  Mise à jour de $poids_{dest} = poids_{dest} + poids(v_i)$
  Déplacement du sommet v_i de V_{ori} vers V_{dest}
  Tantque $poids_{dest} < poids_{moy}$ **faire**
    Mise à jour de V_{adj}
    Sélection du $v_j \in V_{adj}$ de gain maximum
    Mise à jour de $poids_{dest} = poids_{dest} + poids(v_j)$
    Déplacement du sommet v_j de V_{ori} vers V_{dest}
    Suppression de v_j dans V_{adj}
  Fin Tantque
  Sorties: V_{ori}, V_{dest}

Les auteurs Karipis et Kumar considèrent cette méthode comme étant plus efficace que la GGP car elle permet une réduction du coût de coupe et un respect de l'équilibre de la partition au cours de son exécution. Cependant, ces méthodes restent toutes deux fortement dépendantes du choix du sommet de départ. Pour résoudre ce problème, les auteurs proposent de réaliser 4 tirages

aléatoires distincts et de conserver la meilleure solution pour réduire l'aspect aléatoire de cette méthode. Cependant, nous verrons dans la section 2.2.2 que dans la plupart des cas, ce nombre de tirage est loin d'être suffisant pour garantir la convergence de l'algorithme vers une solution proche de l'optimum.

Même si cette méthode est plus efficace que la GGP, elle reste fortement dépendante de la taille du graphe. Pour palier ce problème, les auteurs proposent de combiner cette méthode à une approche multi-niveaux.

1.5 La méthode multi-niveaux

La forme et la taille des graphes sont souvent un frein à la résolution des problèmes de partitionnement par application des méthodes présentées en section 1.4. Prenons par exemple la méthode spectrale, où sa limite principale réside dans la taille du graphe à partitionner. Car si celui atteint une taille trop volumineuse, les matrices nécessaires à sa résolution deviennent lourdes à manipuler. Il en va de même pour les méthodes d'expansion de région qui sont efficaces pour des graphes de petites tailles. Il est donc nécessaire de réduire la taille du graphe, sans pour autant le dénaturer, afin de pouvoir appliquer l'une des ces méthodes.

La méthode multi-niveaux, parallèlement utilisée pour la première fois par Stephen Barnard et Horst Simon dans [BS94] et par Thang Bui et Curt Jones dans [BJ93], offre une solution adéquate à cette problématique de réduction de taille. Son principe repose sur la réduction de la taille du graphe par agrégation des sommets qui le compose. Ainsi, cela revient à manipuler un groupe de sommets au lieu d'un seul tout en conservant la nature initiale du graphe. La sous section suivante présente le mécanisme de fonctionnement de la méthode multi-niveaux.

1.5.1 La structure du multi-niveaux

La méthode multi-niveaux repose sur trois phases distinctes qui ont chacune une action particulière sur le graphe G :

- **Contraction :** la phase de contraction a pour but de réduire la taille du graphe, sans pour autant en affecter la nature, par un processus d'agrégation de sommets. Cette phase, de nature itérative, regroupe des paires de sommets du graphe G tout en conservant une structure similaire. Le graphe G' qui en résulte est de plus petite taille que le graphe originel G. Ainsi, par itération successive, une famille de graphes $\{G_1, ..., G_n\}$, avec $G_1 = G$ est créée, telle que pour chaque graphe G_{i+1}, ces sommets représentent un groupe de sommets du graphe G_i précédent. Le processus s'arrête lorsque le graphe est suffisamment petit pour lui appliquer une méthode de partitionnement de graphes.

- **Partitionnement :** cette phase permet de créer une partition P_k^n du graphe contracté G_n (le dernier de la base de graphe). Le graphe G_n étant de petite taille, il est plus facile et efficace d'appliquer les méthodes présentées en section 1.4 que sur le graphe G d'origine.

- **Affinage :** cette phase consiste à projeter la partition obtenue à partir du graphe G_n sur chacun des graphes de la base de graphe $\{G_1, ..., G_n\}$ jusqu'à obtenir le partitionnement

du graphe $G_1 = G$. Cependant, cette étape nécessite un "recadrage", car une partition globalement bonne sur G_n, peut ne pas être localement bonne sur G_{n-1}. Pour conserver cette qualité, il est nécessaire d'utiliser un algorithme d'affinage local (voir 1.5.3) après chaque projection. Ainsi la qualité de la partition P_k sur G sera très proche de celle obtenue sur G_n.

Algorithme 9 Algorithme multi-niveaux pour le partitionnement de graphes

PROCÉDURE: Multi-niveaux
ENTRÉES: $G = (V,E)$, k nombre de parties, $taille_{max}$ taille souhaitée pour la contraction
  $i = 1$
  Tantque $taille(G_i) > taille_{max}$ **faire**
    G_{i+1} = Contraction(G_i)
    $i = i + 1$
  Fin Tantque
  P_k^n = Partitionnement(G_n)
  Pour i allant de n à 1 **faire**
    P_k^{i-1} = Projection(P_k^i,G_{i-1})
    P_k^{i-1} = Affinage(P_k^{i-1}) par un algorithme d'affinage local
  Fin pour
  $P_k = P_k^1$
  Sorties: P_k

La figure 1.14 et l'algorithme 9 présentent le mécanisme de fonctionnement de la méthode multi-niveaux. L'étape de contraction de celle-ci va permettre d'obtenir un aperçu global du graphe. Ainsi, même si la phase de partitionnement utilise un algorithme local de partitionnement, le résultat sera globalement de bonne qualité pour le graphe d'origine, mais certainement localement mauvais. La phase d'affinage va permettre de l'améliorer localement, tout en conservant la structure globalement bonne de la partition.

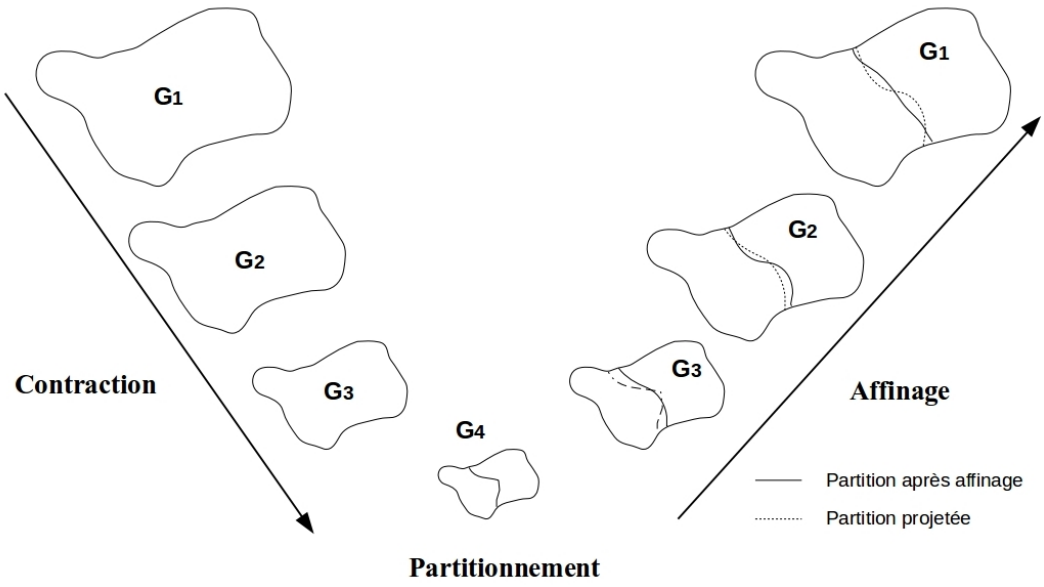


FIGURE 1.14 – Schématisation de la méthode multi-niveaux

1.5.2 Méthodes de contraction

La méthode multi-niveaux fut en premier lieu utilisée pour accélérer les méthodes de partitionnement, qui sont généralement lourdes en calculs et nécessitent un nombre d'itérations proportionnel à la taille du graphe. Cette accélération repose majoritairement sur la phase de contraction qui permet la manipulation de graphe de taille réduite, et donc une meilleure convergence des algorithmes de partitionnement. Cette convergence repose principalement sur la qualité de la contraction, et donc sur la qualité des algorithmes utilisés.

Algorithme 10 Algorithme de contraction de sommets d'Hendrickson-Leland

PROCÉDURE: Contraction_HL
ENTRÉES: $G = (V,E)$, $taille_{max}$ taille souhaitée pour la contraction
  Création d'un base de graphe $Base_G$
  Ajout de $G = G_1$ à la base de graphe $Base_G$
  $i = 1$
  Tantque $taille(G_i) > taille_{max}$ **faire**
    G'_i = Copie(G_i)
    Tantque des arêtes sont disponibles dans G'_i **faire**
      Tirage aléatoire d'une arête non verrouillée $e \in E'_i$ tel que $e = (v_1, v_2)$
      Suppression de e dans E'_i
      Fusion des poids : $poids(v_1) = poids(v_1) + poids(v_2)$
      Pour chaque arrête adjacente à v_2 (e_2) **faire**
        Si si elle possède une extrémité commune avec une des arêtes adjacentes à v_1 (e_1) **Alors**
          Fusion des poids : $poids(e_1) = poids(e_1) + poids(e_2)$
          Suppression de e_2
        Sinon
          Création d'une nouvelle arête reliant v_1 à l'extrémité de e_2
          Suppression de e_2
        Finsi
      Fin pour
      Verrouillage des arêtes adjacentes à v_1
      Suppression du sommet v_2 dans V'_i
    Fin Tantque
    $G_{i+1} = G'_i$
    Ajout de G_{i+1} à la base de graphe $Base_G$
    $i = i + 1$
  Fin Tantque
  Sorties: $Base_G = \{G_1, ..., G_n\}$

Les méthodes de partitionnement ont toutes en commun de chercher à minimiser, avec plus ou moins d'importance, une fonction objectif généralement dérivée du coût de coupe présenté en 1.3.3. Ce qui veut dire, d'un point de vue pratique, que l'on évite au maximum le découpage d'arcs de poids fort lors de la création d'une partition. Une méthode de contraction est efficace si elle permet d'anticiper la minimisation de ces fonctions objectifs lors de l'appariement des sommets. Or, pour éviter de sectionner un arc de poids fort, il serait judicieux de dissimuler son existence à l'algorithme de partitionnement de graphe. C'est ce que propose la phase de contraction de la méthode multi-niveaux, en plus de réduire le nombre de sommets d'un graphe, elle permet de

dissimuler les arcs de poids forts en fusionnant les sommets qui les composent. Ainsi, une base de graphe $\{G_1,...,G_n\}$ est créée tel que G_n possède un minimum d'arcs de poids forts tout en conservant l'information dans les G_i $i < n$ qui le précède. Pour pouvoir respecter l'équilibre de la partition au cours des phases qui suivent la contraction, il est nécessaire de conserver l'information contenue par le poids des sommets avant leur appariement. Pour cela, le poids du sommet $v \in G_{i+1}$ sera égal à la somme des poids des sommets appariés dans G_i. L'application d'une méthode de partitionnement sur un tel graphe garantit une minimisation du coût de coupe et permet une convergence rapide de l'algorithme.

Algorithme 11 Algorithme de contraction HEM

PROCÉDURE: Contraction_HEM
ENTRÉES: $G = (V,E)$, $taille_{max}$ taille souhaitée pour la contraction
  Création d'un base de graphe $Base_G$
  Ajout de $G = G_1$ à la base de graphe $Base_G$
  $i = 1$
  Tantque $taille(G_i) > taille_{max}$ **faire**
    G'_i = Copie(G_i)
    Tantque des sommets sont disponibles dans G'_i **faire**
      Tirage aléatoire d'un sommet non verrouillé $v_1 \in V'_i$
      Sélection de l'arc adjacent à v_1 de poids maximum, $e = (v_1, v_2)$
      Suppression de e dans G'_i
      Fusion des poids : $poids(v_1) = poids(v_1) + poids(v_2)$
      Pour chaque arrête adjacente à v_2 (e_2) **faire**
        Si elle possède une extrémité commune avec une des arêtes adjacentes à v_1 (e_1) **Alors**
          Fusion des poids : $poids(e_1) = poids(e_1) + poids(e_2)$
          Suppression de e_2
        Sinon
          Création d'une nouvelle arête reliant v_1 à l'extrémité de e_2
          Suppression de e_2
        Finsi
      Fin pour
      Suppression du sommet v_2 dans G'_i
    Fin Tantque
    $G_{i+1} = G'_i$
    Ajout de G_{i+1} à la base de graphe $Base_G$
    $i = i + 1$
  Fin Tantque
  Sorties: $Base_G = \{G_1,...,G_n\}$

En effet, si l'appariement de sommets est de mauvaise qualité, quelle que soit la qualité de l'outil de partitionnement utilisé, il en résultera une partition médiocre au sens de la fonction objectif. Pour éviter cela, un certain nombre de méthodes de contraction ont été créées, comme par exemple la méthode de contraction de sommets d'Hendrickson-Leland présentée par l'algorithme 10. Cette méthode repose sur un processus itératif aléatoire, où à chaque itération une arête est prise au hasard dans le graphe $G_i = (V_i, E_i)$ et en est retirée, ainsi que toutes ses arêtes adjacentes. Le sommet résultant de la réunion des deux sommets de l'arête choisie est ajouté à G_{i+1} , ainsi

que les arêtes correspondantes à celles retirées de G_i. Le processus est ré-itéré jusqu'à obtenir un graphe G_n de la taille souhaitée.

Il existe une autre méthode de contraction basée sur le même principe, mais celle-ci utilise en plus les poids des arêtes du graphe. Elle se nomme Heavy Edge Matching (HEM) et a été introduite dans [KK98a]. Le but de celle-ci est de trouver un appariement maximal du graphe qui minimise le coût de coupe. Or, plus grand est l'appariement, plus faible est le poids des arêtes du graphes, ce qui réduit forcement le coût de coupe [KK95]. La méthode HEM consiste à sélectionner aléatoirement un sommet parmi les sommets non verrouillés du graphe G_i, puis à sélectionner l'arête adjacente de poids maximal. Les deux sommets de cette arête forment un nouveau sommet de $G_i + 1$. La suite du processus est équivalente à la méthode de contraction de graphe d'Hendrickson-Leland présentée ci-dessus. L'algorithme 11, présente le fonctionnement de cette méthode de manière détaillée.

Il existe une amélioration de cette méthode pour en réduire l'aspect aléatoire dû au tirage du sommet de départ, elle se nomme Sorted Heavy Edge Matching (SHEM) et est présentée dans [KK98b]. La méthode SHEM trie les sommets en fonction de leurs degrés avant d'appliquer la méthode HEM sur cette liste triée. Elle a l'avantage d'offrir un nombre de solutions limité et de meilleures qualités par rapport à une simple méthode HEM.

1.5.3 Méthode d'affinage

Partant d'une partition P_k^n optimale, une simple projection de celle-ci sur les graphes de la base de graphe ne garantit pas une conservation de la qualité. Le graphe contracté G_n offre une vision globale du graphe G, et contient de l'information "cachée" au sein de ses nœuds. Une solution globalement bonne sur G_{i+1} peut ne pas être localement bonne sur G_i, car celui-ci comporte des informations sur les arcs dont G_{i+1} n'avait pas connaissance. Afin de conserver la qualité locale de la partition, il est donc nécessaire de prendre en compte cette information et de modifier la partition P_k^{i-1} en conséquence. Cette opération se fait à l'aide d'algorithmes d'optimisation locale, dont le but est d'améliorer le coût de coupe de $P_k^j \; \forall j \in \{1, \cdots, n\}$ après chaque projection.

De nombreuses méthodes d'affinage existent. Cependant, tous les algorithmes d'affinage ont en commun deux particularités : leur recherche doit être locale, et doit partir d'une partition existante. Beaucoup de ces méthodes sont basées sur l'algorithme de *Kernighan-Lin* présenté dans [KL70] mais d'autres méthodes existent, comme par exemple la méthode du recuit simulé (voir section 1.4.2) qui peut être utilisé comme une méthode d'affinage.

L'algorithme de Kernighan-Lin permet d'affiner une bissection de graphes en échangeant deux sous-ensembles de sommets de même poids de façon à réduire le coût de coupe de la bissection. L'algorithme fonctionne comme suit : partant d'une bissection existante, l'algorithme échange successivement deux sous-ensembles de la bissection jusqu'à ce que plus aucun sous-ensemble diminuant le coût de coupe ne puisse être trouvé. La sélection des sommets échangeables repose sur les notions de coût intérieur , coût extérieur et différence de coût introduites Kernighan et Lin . Le coût intérieur $Int(v)$ d'un sommet v de la partie V_i de la bissection correspond à la somme des

poids des arcs adjacents à v dont l'autre extrémité se trouve également dans V_i

$$Int(v) = \sum_{v' \in V_i} poids(v, v')$$

Le coût extérieur $Ext(v)$ d'un sommet v de V_i correspond à la somme des poids des arcs adjacents à v dont la seconde extrémité n'appartenant pas à V_i :

$$Ext(v) = \sum_{v' \in V - V_i} poids(v, v')$$

La différence entre le coût extérieur et le coût intérieur du sommet v se nomme :

$$Diff(v) = Ext(v) - Int(v)$$

La différence de coût $Diff(v) \in \mathbb{R}$ est un indicateur sur le gain potentiel que pourrait engendrer le déplacement de v vers la partie voisine. $Diff(v)$ s'interprète comme suit :

$$\begin{cases} Diff(v) < 0 \text{ si v n'est pas sur la frontière de} V_i \\ Diff(v) = 0 \text{ si v est sur la frontière de } V_i \text{ mais son déplacement n'engendre pas de gain} \\ Diff(v) > 0 \text{ si v est sur la frontière de } V_i \text{ et son déplacement engendre un gain} \end{cases}$$

La méthode d'affinage de Kernighan-Lin consiste à créer deux ensembles D_1 et D_2 qui regroupent respectivement les sommets appartenant à V_1 et V_2 dont $Diff(v) > 0$. Pour conserver la balance, il est nécessaire que les ensembles D_1 et D_2 se composent de sommets de même poids. Si tel est le cas, les sommets contenus D_1 sont échangés avec ceux de D_2. Cette méthode d'affinage possède plusieurs versions améliorées tel que l'implémentation de Fiduccia-Mattheyses présenté dans [FM82].

Nous verrons par la suite qu'il est difficile d'appliquer cette méthode d'affinage au k-partitionnement. C'est pourquoi, nous avons implémenté notre propre méthode d'affinage reposant sur une variante de la différence de coupe de Kernighan-Lin adaptée au k-partitionnement. Cette méthode, présentée en section 2.3.2, reprend l'idée de transfert de sommets entre les parties de la partition, mais de façon individuelle.

1.5.4 L'équilibrage de charge

En fonction de la connexité du graphe, et de la nature itérative des méthodes de bissections, il n'est pas toujours possible de créer une partition équilibrée. Or, cette contrainte est fondamentale pour la résolution du problème de partitionnement contraint. Dans ce cas, il est nécessaire de faire appel à un algorithme d'équilibrage de charge pour respecter la balance de partitionnement, même si cela est au détriment du coût de coupe.

Charles Edmond Bichot propose une méthode d'équilibrage de charge dans son livre [BS10]. Celle-ci consiste à déplacer itérativement des sommets se trouvant exclusivement sur la frontière d'une partie, dans le but de réduire le temps d'exécution de l'algorithme. Son mécanisme repose sur la différence de coupe présentée en section 1.5.3, toute partie dont le poids est supérieur au poids moyen calcule le $Diff(v)$ pour chaque $v \in V_i$. Les sommets de chaque partie sont classés par

ordre décroissant de leur $Diff(v)$ et sont déplacés vers une partie voisine à condition que le poids de celle-ci soit toujours inférieur au poids moyen et jusqu'à ce que le poids de la partie courante soit inférieur au poids moyen. Le calcul du poids de chaque partie est répété à chaque "passes" de l'algorithme. Si un déséquilibre important persiste entre les parties, le processus est réitéré. L'algorithme s'arrête lorsque la partition est équilibrée ou lorsque aucune modification n'a eu lieu durant plusieurs "passes".

Il est évident que ce type d'algorithme n'est utilisé qu'en cas d'extrême nécessité. Une bonne implémentation des algorithmes des différentes phases du multi-niveaux , ainsi qu'une bonne convergence des algorithmes de partitionnement évite en général la nécessité d'y avoir recourt. Il est cependant important de les présenter et de les implémenter par sécurité.

Conclusion

Ce chapitre présente dans sa globalité le formalisme de simulation utilisé dans le cadre de notre recherche. L'objectif de cette thèse est d'apporter une optimisation, en terme de temps, aux simulations DEVS dans un cadre distribué. Pour cela, il est nécessaire d'optimiser la structure hiérarchique des modèles DEVS à l'aide d'outils de partitionnement de graphe présentés en détail dans cette section. La littérature offre un grand nombre de méthodes pour résoudre les deux types de problèmes de partitionnement de graphes. L'important est d'être capable de déterminer la nature du problème de partitionnement garantissant la meilleure restructuration de la hiérarchie de modèles.

Même si les méthodes de partitionnement qu'offre la littérature ont toutes fait leurs preuves, il est toujours possible d'apporter quelques améliorations pour garantir l'obtention d'une solution de très bonne qualité. Comme nous avons pu l'observer au cours de ce chapitre, bon nombre de méthodes sont appliquées itérativement pour répondre au problème de k-partitionnement. Or dans ce contexte, une simple application répétée de celles-ci ne garantit en rien une convergence vers l'optimum. C'est pourquoi, nous avons mis un point d'honneur à réduire l'aspect aléatoire de certaines méthodes et avons proposé des améliorations aux méthodes existantes en ce qui concerne leur paramétrisation ou leur implémentation.

La méthode multi-niveaux présentée dans ce chapitre repose principalement sur des algorithmes destinés à la bissection de graphe. Or, dans notre cas, nous sommes soumis au problème du k-partitionnement. Les méthodes telles qu'elles sont conçues sont soit inefficaces, soit trop coûteuses pour résoudre ce genre de problème. Nous nous sommes donc attachés à adapter les algorithmes existants pour les rendre efficaces pour notre problème de partitionnement. Le chapitre qui suit présente l'utilisation des méthodes de partitionnement pour répondre au problème d'optimisation des simulations distribuées DEVS, ainsi que les améliorations et les adaptations que nous avons réalisées pour garantir un partitionnement optimal.

Chapitre 2

Optimisation de la structure hiérarchique DEVS pour une distribution efficace et autonome

Une simulation DEVS distribuée est optimale à condition qu'un maximum de modèles soient exécutables simultanément, que leurs temps d'exécution soient relativement équivalents et que le nombre de messages en transit entre les différents nœuds de calcul soit le plus faible possible. L'optimalité d'une simulation distribuée DEVS repose sur la taille des ensembles de modèles dont la date d'activation est imminente (IMM). Or, pour être optimal, une simulation doit posséder un grand nombre de modèles au sein de ces IMM, à chaque pas de temps, dans le but d'en exécuter un maximum simultanément. De plus, pour être parfaitement optimal à chaque pas de temps, il est nécessaire que la charge de calculs contenue par les modèles de chaque IMM soit équivalente.

Dans un contexte pessimiste, il est rare d'obtenir des IMM de grandes tailles de part la nature réelle du t_a. Afin de montrer l'efficacité des méthodes de partitionnement pour créer une distribution efficace des modèles, nous partons du postula que chaque modèle qui compose la simulation possède un t_a à valeur entière. Cette hypothèse garantit une simulation synchrone et par conséquence, des IMM de grandes tailles. Cette hypothèse forte est utilisée uniquement à titre de validation de nos algorithmes, car dans la réalité, il existe des moyens permettant de manipuler des IMM de grandes tailles sans se contraindre aux simulations purement synchrones (en optant pour un contexte optimiste).

L'objectif de cette section est de présenter, en premier lieu, notre approche pour optimiser la structure hiérarchique des modèles DEVS dans une démarche d'optimisation des simulations distribuées. Cette étape passe par une restructuration de la hiérarchie à l'aide des méthodes de partitionnement de graphes présentées en section 1.3. Dans un second temps, cette section présente l'ensemble des améliorations et des modifications apportées aux méthodes de partitionnement de la littérature dans le but de garantir, à moindre frais, une solution de bonne qualité. Toutes ces améliorations feront l'objet d'illustrations ou de démonstrations afin de parfaitement démontrer soit les insuffisances des méthodes actuelles soit l'adaptation au contexte DEVS.

D'autre part, la méthode présentée dans ce chapitre s'ancre dans un contexte de simulation

distribuée pessimiste mais les grands résultats obtenus peuvent sans souci s'appliquer au cas optimiste.

2.1 Optimisation de la structure hierarchique DEVS

Pour être considérée comme parfaite, une simulation distribuée à k clusters doit, en théorie, permettre une division des temps de calculs par k. Or, dans la pratique, ce speedup est rarement atteignable à cause des coûts informatiques liés à la distribution et aux échanges de messages entre les noeuds de calcul. C'est pourquoi, il est important de créer des clusters homogènes et dont les modèles communiquent le moins possible avec ceux des clusters voisins. Même si le réseau informatique permet de gérer convenablement une certaine quantité de données, ce transfert de message répétitif engendre une perte de vitesse qui est défavorable au speedup.

L'optimisation des simulations DEVS distribuées repose, en premier lieu, sur cette capacité à répartir équitablement les modèles au sein des clusters et en second lieu, sur la capacité à "anticiper" les modèles contenus dans le IMM à chaque pas de temps. Hormis le modélisateur qui a une parfaite connaissance de son modèle, il est rarement possible de déterminer intuitivement cette répartition optimale des modèles. Dans cette thèse, nous proposons de combiner la théorie des graphes à la simulation distribuée pour parvenir à déterminer automatiquement cette répartition optimale des modèles. L'idée maîtresse de notre démarche consiste à répartir équitablement l'intégralité des modèles atomiques parmi les k clusters avant le début de la simulation. Comme le coût de migration d'un modèle d'un cluster à l'autre est trop important, il nécessaire de réaliser une répartition qui soit efficace pour la majorité des pas de temps de la simulation.

2.1.1 Restructuration de la hiérarchie de modèle

Partant d'une hiérarchique de modèles, nous avons vu en section 1.2.5 qu'il est possible de mettre à plat cette hiérarchie grâce aux propriétés de fermeture sous couplage. Cette mise à plat donne accès à une vue globale de la simulation, c'est à dire qu'à partir de celle-ci nous sommes capables de connaître les liens qui unissent chaque modèle atomique. Le résultat qui en découle peut être perçu comme un graphe de modèles, illustrée par la figure 2.1.

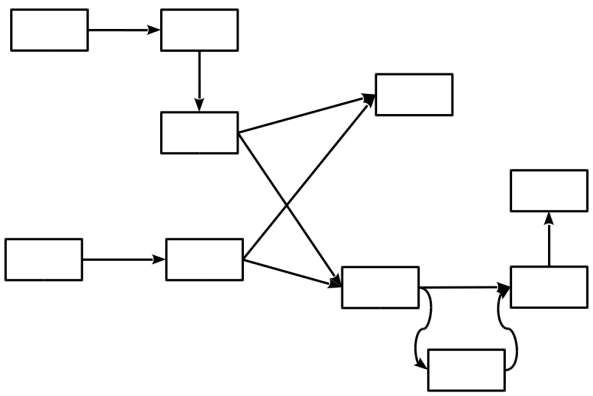


FIGURE 2.1 – Graphe de modèles issu de la mise à plat de la structure hiérarchique d'un modèle DEVS

Notre démarche d'optimisation consiste à recréer une hiérarchie optimale de modèles pour la simulation distribuée, à partir du graphe de modèle issu de la mise à plat. Cette nouvelle hiérarchie comporte trois niveaux :

- le premier niveau se compose d'un coordinateur parent en charge de remonter la date d'activation des prochains modèles atomiques au coordinateur racine, après avoir récolté l'information auprès de ses enfants.
- le second niveau se compose de k coordinateurs, où k correspond au nombre de clusters disponibles. Chaque coordinateur a pour tâche de récolter la date d'activation des prochains modèles atomiques parmi tous les modèles atomiques qui lui sont rattachés.
- le troisième niveau se compose d'ensembles de modèles atomiques repartis équitablement parmi les k coordinateurs du niveau précèdent.

La figure 2.2 présente un exemple de structure hiérarchique optimale pour la simulation distribuée sur 3 clusters. La création de cette nouvelle hiérarchie nécessite une répartition des modèles atomiques en k groupes distincts de façon à satisfaire au mieux les critères suivants :

- répartition équitable des modèles atomiques parmi les k clusters disponibles pour la distribution.
- classer les modèles atomiques de telle sorte que l'échange d'événements entre modèles de cluster différents soit minimal.

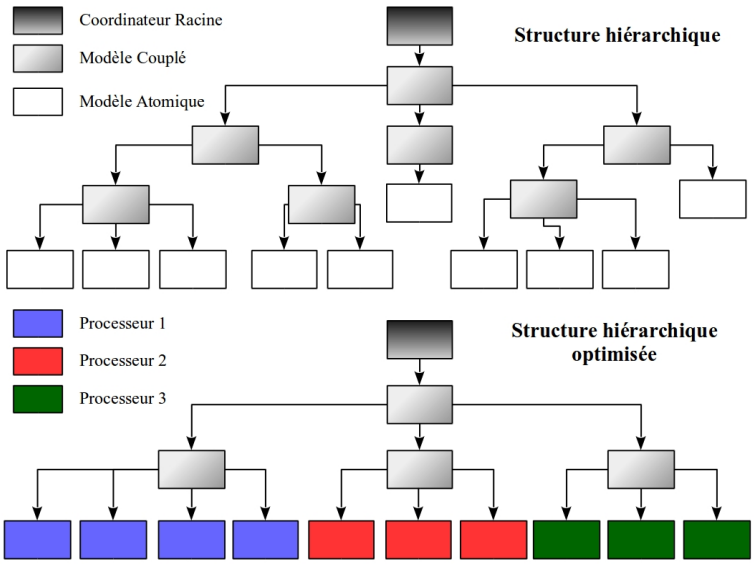


FIGURE 2.2 – Exemple de structure hiérarchique optimale pour la simulation distribuée sur trois clusters

Cette répartition optimale repose sur le partitionnement du graphe de modèles obtenu lors de la mise à plat de la hiérarchie. Afin de respecter au mieux les attentes dûes à la distribution, il est nécessaire de formaliser correctement le problème de partitionnement sous-jacent.

2.1.2 Formalisation du problème de partitionnement

La théorie des graphes ainsi que le partitionnement, qui sont respectivement présentés en section 1.3 et 1.4, offrent des méthodes permettant de partitionner le graphe de modèles de façon à respecter les critères d'optimalité liés aux simulations distribuées. Mais avant de formaliser concrètement le problème de partitionnement attaché à l'optimisation des simulations distribuées, il nécessaire de bien définir la nature ainsi que les spécificités des graphes de modèles.

Type de graphe issu de la mise à plat

Les graphes de modèles issus de la mise à plat de la hiérarchie ont une structure similaire à l'exemple présenté par la figure 2.1. Ces graphes se composent de nœuds, qui sont une représentation abstraite des modèles atomiques, et d'arcs qui représentent l'échange de messages entre ces modèles. Au cours d'une simulation, les modèles atomiques s'échangent des messages suivant une certaine logique liée à la dynamique de la simulation. Il existe donc une dépendance logique entre chaque modèle atomique qui compose le graphe. Ce qui fait de ce graphe, un graphe orienté (voir 1.3.1).

Une simulation peut se composer de différents types de modèles dont la dynamique diffère. Cette dynamique influe le temps d'exécution d'un modèle ainsi que le nombre de messages qu'il produit. Il est donc possible, au cours d'une simulation, que les modèles atomiques qui composent le graphe aient des temps d'exécution différents et une fréquence d'émission de messages différente. Or, ces informations sont primordiales pour la création d'une structure hiérarchique optimale. Il est nécessaire de les prendre en compte lors du partitionnement du graphe de modèles. Pour cela, il est nécessaire de les concaténer au sein du graphe, ce qui en fait un graphe valué (voir 1.3.1) où chaque nœud possède un poids proportionnel au temps d'exécution du modèle auquel il est rattaché et chaque arc possède un poids correspondant à la fréquence d'émission de messages entre les modèles atomiques qui le compose. Ces informations sont dans un premier temps purement théorique, nous sommes partis du postula que chaque modèle possède un même temps d'exécution et que la quantité de messages en transit entre les modèles est équivalente. Le graphe résultant de ce postula est donc unitaire, c'est à dire que l'ensemble de ses poids valent 1. Nous verrons par la suite, que ces hypothèses sont parfois incomplètes voir erronées.

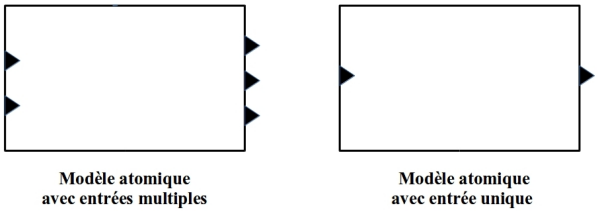


FIGURE 2.3 – Exemple de modèle atomique avec entrées multiples et entrée unique

Le formalisme DEVS offre différentes structures possibles pour un modèle atomique, comme le montre la figure 2.3, il est possible de créer un modèle atomique se composant d'entrées et de sorties multiples, où on associe en général une sortie par destinataire. Cependant, nous avons préféré la seconde formalisation du modèle atomique qui se compose d'un unique port d'entrée et

de sortie. S'il existe plusieurs destinataires, le message contenu par le port de sortie est dupliqué puis distribué à chacun des modèles destinataires. Ce genre de formalisation donne naissance à un graphe de modèles où les arêtes multiples sont interdites. De plus, comme le formalisme DEVS interdit la connexion d'un modèle atomique à lui-même, il est impossible de trouver des boucles au sein du graphe. Ce qui garantit à ce graphe d'être un graphe simple (voir 1.3.1).

Nature du problème de partitionnement

Le partitionnement du graphe de modèle doit satisfaire les critères énoncés en section 2.1.1 pour garantir la création d'une hiérarchie optimale. La priorité recherchée lors de cette phase est de garantir une répartition équilibrée des modèles atomiques entre les clusters. Sachant que chaque sommet du graphe est un modèle atomique, cela revient à créer une partition de sommets équilibrés. Rappelons qu'une partition est équilibrée si sa balance de partitionnement, présentée en section 1.3.3, est inférieure ou égale à 1.05.

Pour éviter de détériorer le gain engendré par cette répartition équitable, il est nécessaire de minimiser le coût de transfert de message entre des modèles atomiques de clusters différents. Ce transfert de message étant symbolisé par les arcs du graphe, cela revient à éviter, au maximum, de couper les arcs de poids forts lors du partitionnement. Ces arcs symbolisent de nombreux échanges de message entre les modèles. Lors d'une distribution, l'échange de message entre les clusters a un coût important qui est lié à l'utilisation du réseau. Plus les transferts sont importants et plus le coût engendré par le réseau l'est aussi. C'est pour cela qu'il est important de réduire au maximum le découpage d'arcs de poids forts. Cette contrainte correspond à la minimisation du coût de coupe présentée en section 1.3.3.

Le problème de partitionnement lié à la création d'une nouvelle structure hiérarchique optimisée se rapporte au problème de partitionnement contraint présenté en section 1.3.4. Ce choix se justifie par l'ordre de priorité des contraintes à respecter. Lors du partitionnement, l'objectif principal est de créer une partition équilibrée tout en minimisant le coût de coupe. Le respect de la balance sera considéré comme notre contrainte "forte" et la minimisation du coût de coupe comme la contrainte "faible". Ceci implique que la partition doit être équilibrée même si cela est au détriment du coût de coupe. Cependant, il est important d'utiliser une approche conciliant au mieux les deux contraintes pour garantir une simulation optimale en temps.

2.1.3 Choix des méthodes de partitionnement

Face aux simulations de plus en plus grandes en nombre de modèles, il devient difficile d'employer des méthodes de partitionnement directes sur des graphes comportant au minimum des dizaines voir des centaines de milliers de modèles. Pour certaines, le résultat n'en serait que médiocre, sans considérer le temps engendré par l'application de ces algorithmes.

Pour pallier ce problème, la littérature offre la méthode Multi-niveaux, présentée en section 1.5. Cette méthode, longtemps considérée comme une approche d'accélération des méthodes de partitionnement existantes, est appréciée pour son efficacité tant sur le plan théorique que pratique. Elle possède le double avantage d'offrir une partition de qualité, tout en réduisant considérable-

ment le temps lié au partitionnement.

Il est donc naturel que notre choix se porte sur ce type de méthode. Comme présenté en section 1.5.1, cette méthode repose sur trois phases pouvant être réalisées à partir de différents algorithmes. Il a donc été nécessaire de paramétrer cette méthode Multi-niveaux pour répondre au problème de partitionnement contraint, sous-jacent au problème d'optimisation de la structure hiérarchique de modèles DEVS. Notre paramétrisation est la suivante :

- **Contraction** : dans un premier temps, nous avons opté pour l'utilisation des méthodes HEM et SHEM, présentées en section 1.5.2. Nous verrons dans la section 2.3 que même si ces méthodes offrent une contraction de bonne qualité, elles ont le désavantage de reposer sur un mécanisme aléatoire. Ce qui implique, que d'un Multi-niveaux à l'autre, la contraction n'est pas la même et que par conséquent la qualité de la partition peut en être affectée. Nous avons donc proposé notre propre méthode de contraction, proche de celles énoncées dans la section 1.5.2, mais supprimant l'aspect aléatoire présent dans celles-ci. Cette méthode est présentée en détail dans la section 2.3.1.

- **Partitionnement** : cette phase repose sur deux types de méthodes utilisant chacune une approche totalement différente. La première, et la plus importante, est la méthode Greedy Graph Growing Partitioning (GGGP) présentée en section 1.4.4. Le choix de cette méthode n'est pas anodin, son processus de partitionnement repose sur un mécanisme de voisinage cherchant à créer une bissection parfaitement équilibrée. Or, dans notre contexte de partitionnement contraint, cette recherche de l'équilibre parfait est très appréciée. Il est donc naturel de s'intéresser tout particulièrement à cette méthode. Cependant, nous verrons dans la section 2.2 que cette méthode théoriquement très avantageuse comporte quelques failles, que nous avons cherché à contourner dans le cadre de cette thèse. Cette section présente différentes optimisations de cette méthode pour garantir, à moindre frais, une convergence vers une partition de qualité proche de l'optimum. La seconde méthode à laquelle nous nous sommes intéressés est une méthode spectrale, présentée en section 1.4.3. L'objectif de celle-ci est de pouvoir comparer les résultats obtenus par l'application de la méthode GGGP à une méthode reposant sur les mathématiques formelles.

- **Affinage** : pour cette phase, nous avons créé un algorithme d'optimisation locale basé sur la différence de coût de Kernighan-Lin, présentée en section 1.5.3. Le processus d'affinage cherche, en premier lieu, à respecter la balance de partitionnement. Or, si par malheur, celle-ci n'a pu être respectée lors de la phase de partitionnement, il est nécessaire d'appliquer un algorithme d'équilibrage de charge en amont de l'algorithme d'affinage. Cette démarche, ainsi que notre méthode d'affinage, sont toutes deux présentées en section 2.3.2.

En fonction de la méthode de partitionnement employée, le niveau de contraction nécessaire peut varier. Dans le cadre de l'utilisation de la méthode GGGP, il est nécessaire de contracter le graphe de telle sorte que le nombre de modèles qui le compose soit inférieur à 600. Dans un cas parfait, il serait souhaitable que le nombre de modèles ne dépasse pas 200. Cependant, nous verrons dans la section 2.2 qu'il n'est pas toujours possible de respecter cette contrainte de taille en fonction du nombre de parties voulu. Pour l'utilisation de la méthode spectrale, seul le poids des matrices à manipuler est un frein. C'est pourquoi, il est nécessaire de réduire la taille des graphes à

manipuler de façon à réduire la taille des matrices correspondantes. Cette réduction est cependant moins forte que pour la méthode GGGP. Une contraction de l'ordre de 4000 à 1000 nœuds est nécessaire. Nous étudierons l'impact de cette réduction sur la qualité du partitionnement dans la section 4.3.3.

L'optimalité de la simulation distribuée repose en grande partie sur la résolution du problème de partitionnement. Or, quel que soit le problème, la recherche de la solution optimale n'est pas simple et nécessite en général un nombre de calculs important. De plus, l'obtention de cet optimal n'est pas toujours réalisable de part la nature itérative des méthodes de bissection. La recherche de cet optimum, ou d'une solution proche, passe par un ensemble d'améliorations et procédés pour parcourir la majeure partie de l'espace de solution du problème de partitionnement contraint.

2.2 Optimisation de la méthode Greedy Graph Growing Partitioning

Les méthodes d'expansion de région, dont notamment la méthode GGGP (voir section 1.4.4), offrent un bon compromis entre efficacité et rapidité. Cependant, les algorithmes tels qu'ils sont présentés dans la littérature ne permettent pas de gérer convenablement tous les cas de figure. De plus, l'utilisation récursive de ce type de méthode est un frein naturel à leur efficacité. Pour remédier à cela, nous proposons dans le cadre de cette section des améliorations à la méthode GGGP pour combler quelques lacunes des algorithmes actuels.

2.2.1 Amélioration de la méthode GGGP

L'algorithme GGGP (8) tel qu'il est construit actuellement repose exclusivement sur la notion de voisinage. Partant d'un ensemble contenant l'intégralité des sommets du graphe, la méthode commence par tirer au hasard un sommet du graphe et le déplace vers un second ensemble. La mécanique de cette méthode est simple, elle repose sur le déplacement de sommets de l'ensemble de départ vers l'ensemble d'arrivée à condition qu'ils soient adjacents à ceux présents dans l'ensemble d'arrivée. Le processus est répété jusqu'à atteindre la moitié du poids du graphe. Or, dans la pratique, il n'est pas garanti qu'il soit possible de trouver de tels sommets.

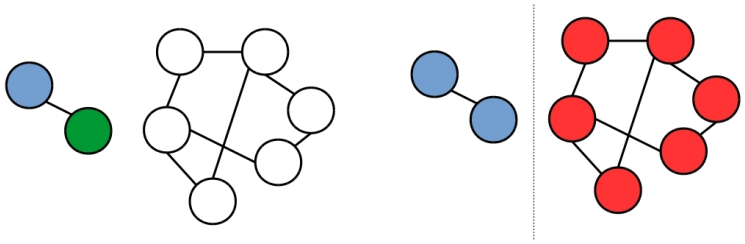


FIGURE 2.4 – Exemple de partition déséquilibrée par application de la méthode GGGP sans amélioration pour les graphes non-connexes

Prenons par exemple, la situation du graphe de la figure 2.4. Dans celle-ci, les sommets en bleu correspondent aux sommets de l'ensemble d'arrivée, les sommets rouges à ceux de l'ensemble de départ et les sommets verts sont les sommets éligibles à un déplacement. La graphe de gauche

montre le tirage aléatoire d'un sommet du graphe (en bleu) et le futur sommet éligible à un déplacement (en vert). Le graphe de droite montre l'évolution de la partition à l'itération suivante. On peut constater, que dans cet exemple, le processus s'arrête par manque de sommet éligible à un déplacement. Or, la partition ne respecte pas la condition d'équilibre de poids des parties car l'algorithme GGGP tel qu'il est actuellement conçu ne possède pas de moyen pour s'extraire de ce type de situation.

Le problème majeur de la méthode GGGP repose sur la connexité (voir section 1.3.1) du graphe à partitionner. Car en effet, si celui-ci est non-connexe il y a de forte probabilité de tomber sur le type de situation illustré par la figure 2.4. Or, rappelons que c'est une méthode de bissection utilisée de manière itérative, ou récursive, pour réaliser un k-partitionnement. Il est donc évident que, dans la plupart des cas, l'application de cette méthode modifie la structure du graphe et le rend non-connexe au cours de son exécution.

Algorithme 12 Algorithme GGGP amélioré

PROCÉDURE: GGGP_amélioré
ENTRÉES: $G = (V,E)$, P_{biais}
  Initialisation $V_{ori} = V$ contient tous les sommets de V
  Initialisation V_{dest} et V_{adj} sont vides
  Initialisation $poids_{dest} = 0$ du poids de l'ensemble destination à 0
  Calcul du poids moyen $poids_{moy} = \frac{poids(V)}{2}$
  Tirage aléatoire d'un sommet $v_i \in V_{ori}$
  Mise à jour de $poids_{dest} = poids_{dest} + poids(v_i)$
  Déplacement du sommet v_i de V_{ori} vers V_{dest}
  Tantque $poids_{dest} < poids_{moy}$ **faire**
    Mise à jour de V_{adj}
    Si V_{adj} est vide et $poids_{dest} < P_{biais}$ % de $poids_{moy}$ **Alors**
      Tirage du sommet $v_i \in V_{ori}$ de degré minimum
      Mise à jour de $poids_{dest} = poids_{dest} + poids(v_i)$
      Déplacement du sommet v_i de V_{ori} vers V_{dest}
      Mise à jour de V_{adj}
    Finsi
    Sélection du $v_j \in V_{adj}$ de gain maximum
    Mise à jour de $poids_{dest} = poids_{dest} + poids(v_j)$
    Déplacement du sommet v_j de V_{ori} vers V_{dest}
    Suppression de v_j dans V_{adj}
  Fin Tantque
  Sorties: V_{ori}, V_{dest}

Pour contourner ce problème, nous avons proposé une amélioration de la méthode GGGP. Elle consiste à autoriser le tirage d'un sommet dans l'ensemble de départ si aucun autre sommet n'est éligible et si la contrainte de poids n'est pas respectée avec un certain "biais". Cette notion de "biais" laisse libre choix à l'utilisateur de vouloir, ou non, respecter la balance de partitionnement. Car, rappelons le, cette méthode peut être également utilisée pour le problème du partitionnement non contraint. Une fois que le tirage du nouveau sommet est réalisé, le processus de la méthode GGGP reprend normalement. Cette approche a pour intérêt de forcer l'équilibre de la partition

mais ceci au détriment du coût de coupe. Cependant, afin d'en minimiser l'impact, nous proposons de sélectionner le sommet de degré minimum.

La figure 2.5 présente la continuation du processus de partitionnement de l'exemple ci-dessus par application de l'amélioration de la méthode GGGP. On peut constater que, grâce à cette amélioration, la partition est équilibrée. L'algorithme 12, présente l'amélioration apportée à l'algorithme GGGP pour garantir le respect de l'équilibre de la partition en cas de besoin.

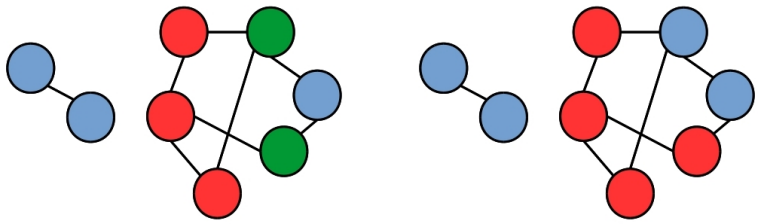


FIGURE 2.5 – Équilibrage de la partition par application de la méthode GGGP améliorée

Pour un paramétrage de P_{biais} à 100, la méthode GGGP améliorée garantit l'obtention d'une partition équilibrée, ce qui résout le problème lié à la contrainte "forte" de notre problème de partitionnement. Cependant, l'aspect aléatoire du tirage du sommet de départ de cette méthode rend la résolution de la contrainte "faible" compliquée. La minimisation du coût de coupe est totalement dépendant de ce tirage aléatoire.

2.2.2 Réduction de l'aspect aléatoire pour une recherche fixe de l'optimum

La méthode GGGP, comme bon nombre de méthode d'expansion de région, est fortement dépendante du tirage du sommet de départ en ce qui concerne l'évolution du coût de coupe. Même si celle-ci n'est pas la contrainte prioritaire de notre problème de partitionnement, il reste cependant très important de la minimiser dans le but d'affecter le moins possible le gain obtenu par l'équilibrage des modèles. Pour réduire l'impact de cet aspect aléatoire, les auteurs de la méthode GGGP propose de réaliser quatre tirages indépendants et de conserver le meilleur résultat. Cependant, nous sommes en droit de nous demander si ces seuls tirages suffisent à garantir, à tous les coups, l'obtention de la bissection de coût de coupe minimale.

Influence du tirage de départ sur la qualité du coût de coupe

Pour réfuter, ou valider, cette hypothèse, nous proposons d'étudier l'impact du choix du sommet de départ sur deux graphes de taille 200 dont la structure diffère. Le premier possède une structure dite "simple", c'est à dire que les nœuds sont relativement peu connectés (au maximum 3 à 5 voisins). Et le second à une structure dite "complexe", où les nœuds sont très fortement connectés (au maximum 10 à 15 voisins). Pour chaque graphe, on réalise 200 bissections par GGGP, avec chacun des sommets du graphe comme sommet de départ. L'objectif de cette démarche est de comptabiliser le nombre de coûts de coupe différents obtenus et dans quelle proportion.

La figure 2.6 présente le résultat obtenu pour un graphe à structure "simple". Pour ce type de graphe, nous pouvons constater que seul trois coûts de coupe différents résultent des 200 bissec-

tions réalisées. Dans ce cas, presque 50% des sommets de départ offrent une bissection de coût de coupe minimale.

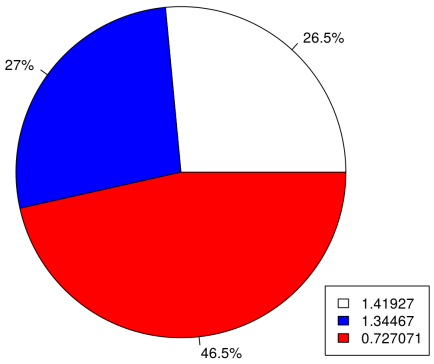


FIGURE 2.6 – Exemple de répartition des sommets en fonction des coûts de coupe pour un graphe de taille 200 à structure "simple"

L'hypothèse faite par les auteurs Kernighan et Lin, suppose que le tirage de seulement 4 sommets de départ garantit l'obtention de l'optimum avec une forte probabilité. Calculons formellement cette probabilité, à l'aide des outils statistiques. Si l'on transpose cette problématique à celle des probabilités, cela revient à obtenir au moins un tirage parmi les quatre réalisés (sans remise et sans tenir compte de l'ordre) dont le résultat est égal au coût de coupe minimum. Si l'on considère que ce problème possède les deux issues suivantes :

- obtenir une bissection de coût de coupe minimum
- obtenir une bissection de coût de coupe supérieur au minimum

Cela revient à résoudre une loi de probabilité hypergéométrique $H(S,n,p)$ définit comme suit :

$$P(X=k) = \frac{C_{S_m}^{k} \times C_{S-S_m}^{n-k}}{C_S^n}$$

Où,

- p : probabilité que le coût de coupe de la bissection soit minimum pour un tirage
- S : nombre de sommets total du graphe
- n : nombre de tirages sans remise $(1 \leq n \leq S)$
- k : nombre de résultats de coût de coupe minimum souhaité parmi les n tirages $(0 \leq k \leq n)$
- S_m : nombre de sommets fournissant un coût de coupe minimum

Pour l'exemple du graphe "simple", illustré par la figure 2.6, nous allons étudier l'évolution de la loi hypergéométrique de paramètre $H(200,n,0.465)$ pour $n = \{4,10,50,100,150\}$ à l'aide du tableau 2.1. Cette démarche va nous permettre de déterminer la probabilité d'obtenir la bissection de coût de coupe minimum en fonction du nombre de tirages initiaux.

Dans ce cas, l'obtention d'une bissection de coût de coupe minimale est garantie dès 4 tirages de sommets de départ. Rappelons que même si ces résultats sont excellents, il existe tout de même un risque de convergence vers une solution moindre par l'aspect aléatoire des tirages.

Nombre de tirages	4	10	50	100	150
$P(Min_{cut})$ en %	92.02	99.84	99.96	99.99	99.99

TABLEAU 2.1 – Évolution de la probabilité d'obtenir une bissection du coût de coupe minimale en fonction du nombre de tirage de sommet de départ pour un graphe "simple"

Réalisons le même test à partir d'un graphe "complexe" de taille 200. La figure 2.7 donne une répartition des sommets de départ en fonction du coût de coupe à l'aide d'un histogramme à gauche et d'une boîte à moustache à droite. Le choix de ces deux représentations est dû au grand nombre de solutions, qui se portent à 23 dans ce cas. En observant l'histogramme, nous pouvons constater que le coût de coupe minimum obtenu par application de 200 bissections est de 4.11, ce qui représente 2% des sommets de départ. De plus, environ 50% des sommets de départ offrent une solution proche de l'optimum. Ces résultats sont plus difficiles à interpréter que ceux du graphe simple.

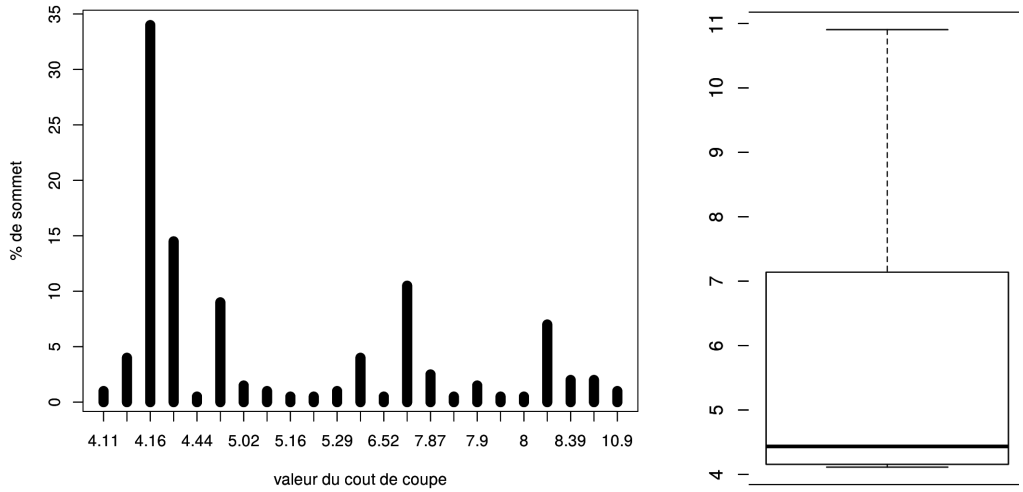


FIGURE 2.7 – Exemple de répartition des sommets en fonction des coûts de coupe pour un graphe de taille 200 à structure "complexe"

En fixant les mêmes hypothèses que précédemment, nous sommes capables d'évaluer la validité de l'hypothèse des auteurs de la méthode GGGP. En partant de l'hypothèse forte que seule la recherche de l'optimum compte, étudions l'évolution de la loi hypergéométrique de paramètre $H(200,n,0.0.02)$ pour $n = \{4,10,50,100,150\}$ à l'aide du tableau 2.2.

Nombre de tirages	4	10	50	100	150
$P(Min_{cut})$ en %	7.82	18.68	68.68	93.93	99.64

TABLEAU 2.2 – Évolution de la probabilité d'obtenir une bissection du coût de coupe minimale en fonction du nombre de tirage de sommet de départ pour un graphe "complexe"

Même si au delà de 100 tirages la probabilité d'obtenir une bissection optimale est supérieure à 93%, ces résultats sont loin d'être suffisants pour garantir une convergence absolue vers l'optimum. Ces résultats permettent d'affirmer qu'en fonction de la forme du graphe, le nombre de

tirages de départ ne doit pas se limiter à 4 pour garantir une convergence vers l'optimum.

En vue des résultats présentés ci-dessus, il semble évident d'affirmer que nous ne pouvons pas nous limiter à 4 tirages de sommets de départ pour garantir une solution de bonne qualité. Ni même à $\frac{n}{2}$ tirages, où n est le nombre de sommets du graphe.

Encapsulation du GGGP dans une structure d'optimisation

La méthode GGGP, en plus d'être une méthode efficace, a l'avantage d'être peu coûteuse en temps à partir du moment où la taille du graphe n'excède pas les 600 sommets. Reprenons l'exemple du graphe "complexe", l'exécution de l'algorithme GGGP pour les 200 sommets de départ n'a demandé que 0.241 secondes, soit une moyenne de 0.0012 seconde pour une bissection de graphe. En vue du peu de temps que demande l'exécution de l'algorithme GGGP et de l'importance de garantir un coût de coupe minimum, il est préférable de tester chaque sommet de départ et de conserver la meilleure bissection.

Les figures 2.6 et 2.7 tendent à montrer qu'un certain nombre de sommets fournissent les mêmes bissections. Rappelons que cette méthode repose exclusivement sur une notion de voisinage, ce qui implique que deux sommets adjacents sont fortement susceptibles de fournir une partition équivalente. Afin de réduire le nombre de bissections, et en partant de l'hypothèse qu'un groupe de sommets proches fournit la même partition, il est possible de mettre en place un système de sélection des sommets de départ. En effet, si l'on considère que tout sommet se trouvant à une distance *dist* du sommet de départ fournit le même coût de coupe, alors il est inutile de réaliser une bissection à partir de ce sommet. Ici, la distance correspond au nombre d'arcs composant le chemin entre deux sommets. Cette démarche permet de réduire le nombre de bissections, sans risquer d'endommager la qualité du coût de coupe. L'algorithme 13 présente le mécanisme de verrouillage des sommets susceptibles de fournir un coût de coupe similaire. Au cours de nos expérimentations, nous nous sommes aperçus qu'au delà d'une distance de 3, cette approche devient défavorable car trop restrictive. En vu de la rapidité initiale de la GGGP, il n'est pas obligatoire d'utiliser cette approche. C'est pourquoi son utilisation est facultative.

Algorithme 13 Algorithme de restriction des tirages par distance

PROCÉDURE: Tirage_distance
ENTRÉES: $G = (V, E)$, *dist*
  Initialisation de $V_{verrou} = \emptyset$ l'ensemble des sommets verrouillés
  Initialisation de $V_{elu} = \emptyset$ l'ensemble des sommets éligibles à un tirage
  Tantque taille(V_{verrou}) $\neq$ taille(V) **faire**
    Tirage d'un sommet dans $v_i \in V$ (sans remise)
    Si $v_i \notin V_{verrou}$ **Alors**
      Tirage au hasard d'un sommet $v_i \in V$
      Ajout à V_{verrou} de tous les sommets dont les chemins les reliant à v_i sont $\leq dist$
      Ajout de v_i à V_{elu} et à V_{verrou}
    Finsi
  Fin Tantque
  Sorties: V_{elu}

Pour garantir la solution optimale d'une bissection, il est nécessaire de tester un maximum de tirages de sommets de départ. Or, nous avons vu ci-dessus que la restriction de ces tirages à un nombre trop limité peut, dans certains cas, être défavorable au coût de coupe. C'est pourquoi nous proposons d'intégrer la méthode GGGP à un algorithme d'optimisation dont le rôle est de piloter, en quelque sorte, le paramétrage de la GGGP. Ce paramétrage consiste à déterminer le sommet de départ qui fournit la meilleure bissection et permet de contrôler l'équilibre de celle-ci. Si celui-ci n'est pas respecté, le paramètre P_{biais} du GGGP amélioré (voir algorithme 12) est modifié de façon à autoriser l'accès au sommet de la partie non connexe du graphe. Comme cette utilisation a pour désavantage d'amplifier la non-connexité des sous-graphes issus de la bissection, et par conséquent d'impacter la qualité du coût de coupe lors d'une utilisation par bissection itérative hiérarchique, nous privilégions une solution équilibrée de coût de coupe moindre obtenue par application du GGGP standard ($P_{biais} = 0$) à celle obtenue par la GGGP améliorée, à condition qu'une telle solution existe.

L'algorithme d'optimisation (14) laisse libre choix à l'utilisateur du nombre de sommets de départ à tester (Nb_t). Comme la GGGP est peu coûteuse en temps, ce paramètre est souvent égal au nombre de sommets du graphe à partitionner. Pour chaque bissection, on vérifie si la balance est respectée et on conserve celle de coût de coupe minimum pour un paramétrage de la GGGP amélioré à $P_{biais} = 0$. Or, dans certains cas, il n'est pas possible de trouver une solution respectant la balance. C'est pourquoi, dans ce cas, on réitère le processus pour un paramétrage de la GGGP améliorée à $P_{biais} = 100$. Cette fois, il est certain de trouver une bissection de balance unitaire, même si cela risque d'impacter le coût de coupe à la prochaine application de la méthode sur le graphe résultant. Il est possible d'incorporer le tirage par distance en ajoutant le paramètre *dist* à l'algorithme d'optimisation.

Algorithme 14 Algorithme d'optimisation encapsulant la méthode GGGP

PROCÉDURE: Optimisation_GGGP
ENTRÉES: $G = (V, E)$, Nb_t, $dist$
  Initialisation $V_{elu} = \emptyset$
  Initialisation $P_{biais} = 0$
  Si $dist \neq 0$ **(*) Alors**
    $V_{elu} =$ **Tirage_distance**(G, $dist$)
  Sinon
    $V_{elu} = V$
  Finsi
  Pour i allant de 1 à Nb_t **faire**
    Tirage aléatoire d'un $v_i \in V_{elu}$ sans remise
    $\{V_{ori}, V_{dest}\}$ **= GGGP_amélioré**(G, P_{biais}, v_i)
  Fin pour
  Conservation de la bissection de coût de coupe minimum
  Si bal($\{V_{ori}, V_{dest}\}$) > 1.05 **Alors**
    $P_{biais} = 100$
    Reprise de l'algorithme à **(*)**
  Finsi
  Sorties: V_{ori}, V_{dest}

L'utilisation de l'algorithme 14, nécessite une légère modification dans l'implémentation de la fonction GGGP. Le sommet de départ de l'algorithme devient un paramètre de celle-ci. L'algorithme 15 présente cette modification.

Algorithme 15 Modification de la GGGP améliorée pour mettre en paramètre le sommet de départ

PROCÉDURE: GGGP_amélioré
ENTRÉES: $G = (V, E)$, P_{biais}, v_{depart}
Même procédé que la GGGP améliorée mais avec un tirage du sommet de départ externe
Sorties: V_{ori}, V_{dest}

L'optimisation de nos méthodes est certes une importante étape pour l'obtention de l'optimum à l'échelle de la bissection, mais son application par itération hiérarchique ne garantit en rien la convergence vers l'optimum global du problème de k-partitionnement.

2.2.3 Optimisation pour le k-partitionnement

Comme nous l'avons vu en section 1.4.4, beaucoup de méthodes initialement conçues pour la bissection de graphes sont généralisables au k-partitionnement par le biais d'une application itérative hiérarchique. Ce type d'approche rend très difficile le parcours total du spectre de solutions du problème de partitionnement, il est donc rare de parvenir à atteindre l'optimum. Cependant, plusieurs stratégies existent pour garantir que l'approche par itération hiérarchique tende vers une solution locale proche de cet optimum. Rappelons que dans le cadre de cette thèse, nous cherchons à déterminer, à moindre frais, une partition équilibrée et de bonne qualité. Il n'est donc pas envisageable de perdre une grosse quantité de temps pour la recherche d'un optimum qui, au final, ne sera probablement que légèrement meilleur.

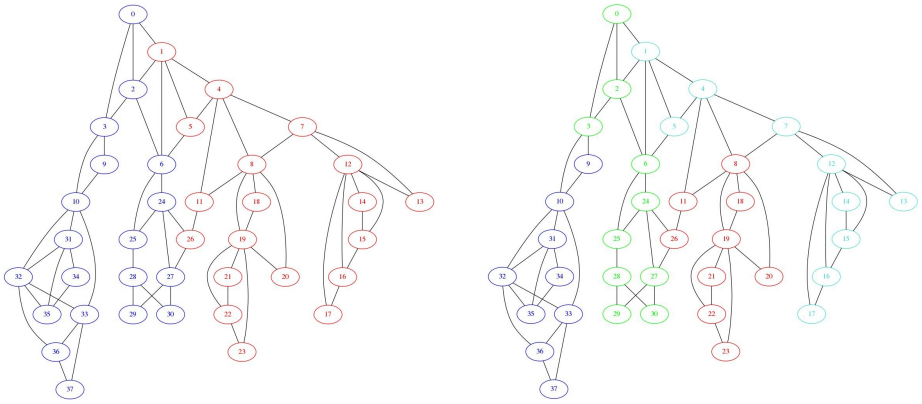
Stratégies pour le k-partitionnement

Notre approche pour résoudre le problème du k-partitionnement par application itérative hiérarchique de la méthode GGGP repose sur deux stratégies. La première, et la plus importante, repose sur une hypothèse forte : *"la recherche de la bissection optimale, à chaque itération, garantit la convergence vers une partition de qualité proche de l'optimum"*. D'un point de vue purement logique, le fait de minimiser le coût de coupe à chaque itération force l'algorithme à exclure les mauvaises solutions, sans pour autant garantir que celle obtenue sera la meilleure. La recherche de la solution optimale, pour chaque bissection, contraint le problème de k-partitionnement et peut avoir pour effet d'empêcher l'accès à certaines solutions. Les figures 2.8 et 2.9 présentent l'évolution d'une partition et de son coût de coupe au fil des itérations.

L'obtention d'une partition en 4 parties s'obtient par application de 3 bissections itératives. Pour la figure 2.8, les partitions sont obtenues par recherche de la meilleure bissection à chaque itération. Les coûts de coupe de ces bissections sont donnés dans le tableau ci-dessous :

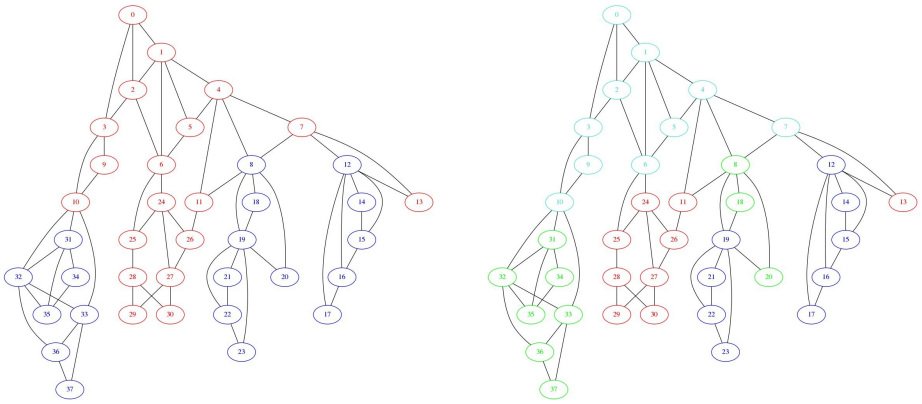
Bissection 1	Bissection 2	Bissection 3	4-Partition
0.7366	0.2371	0.3245	1.2983

On peut observer que, dans ce cas, chaque bissection donne naissance à un sous-graphe connexe. Ce qui n'est pas le cas du 4-partitionnement de la figure 2.9. En effet, dans cet exemple, les bissections retenues ne sont pas celles de coût de coupe minimal.



cut = 0.7366 **cut = 1.2983**

FIGURE 2.8 – Exemple de 4-partitionnement par application de bissection optimale



cut = 0.7705 **cut = 1.6113**

FIGURE 2.9 – Exemple de 4-partitionnement par application de bissection sans recherche de l'optimal

On peut observer que le résultat de la partition n'est pas homogène, les sous-parties de la partition ne sont pas connexes. Ce qui se traduit directement par un coût de coupe de qualité médiocre. Observons l'évolution de ce coût de coupe au fil des bissections :

Bissection 1	Bissection 2	Bissection 3	4-Partition
0.7705	0.3363	0.5045	1.6113

Dans ce cas, le fait d'avoir autorisé la conservation d'une première bissection de coût de coupe supérieur a eu un effet "boule de neige" sur la qualité des bissections suivantes. En effet, on peut constater que les coûts de coupe du deuxième partitionnement sont tous supérieurs à ceux du premier utilisant des bissections optimales.

Ceci nous conforte dans l'idée qu'il est intéressant de réaliser une bissection optimale à chaque itération même si cela peut empêcher l'accès à l'optimum global. Cette approche garantit une solution de bonne qualité, ce qui n'est pas le cas lorsqu'on autorise la conservation d'une bissection

de qualité moindre. Schématiquement, on peut considérer que cette approche garantit l'obtention d'une partition dont le coût de coupe se positionne à un certain pourcentage de l'optimum. En fonction de la structure du graphe et du nombre de partitions souhaité, nous avons estimé que la solution fournit par cette première approche possède un coût de coupe compris entre l'optimum et 15 à 20% maximum de cet optimum. Ce positionnement est illustré par la figure 2.10.

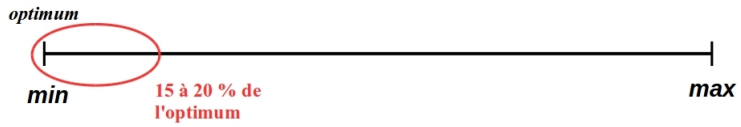


FIGURE 2.10 – Positionnement du coût de coupe obtenu pour le k-partitionnement par bissection itérative optimale par rapport à l'optimum

La seconde stratégie pour réaliser un k-partitionnement prend en compte le fait qu'une bissection de qualité inférieure à l'optimum peut permettre, dans certains cas, l'accès à une solution inaccessible par application de la méthode itérative hiérarchique par bissections optimales. Et dans l'absolu, il est possible que cette solution soit l'optimum. Afin de permettre l'accès à cette catégorie de solutions, dites difficiles, nous proposons une variante de l'algorithme d'optimisation du GGGP. Contrairement à la méthode **Optimisation_GGGP** (14) qui ne conserve que la meilleure solution, cette nouvelle approche autorise la conservation de solutions à condition que celles-ci soient proches de la meilleure précédemment obtenue. Ce mécanisme est inspiré par celui du Recuit-Simulé présenté en section 1.4.2. Cette approche permet de perturber la solution afin de sortir des extremums locaux. Cette stratégie est présentée par l'algorithme 16.

Algorithme 16 Algorithme d'optimisation de la GGGP par perturbation pour le k-partitionnement

PROCÉDURE: Optimisation_GGGP_Perturbation
ENTRÉES: G, Nb_t, P^K ensemble de partitions, *index* de l'ensemble à partitionner
$PP = \{P^1, P^2, \cdots, P^m\}$ où $P^i = \{V_1, V_2, \cdots, V_k\}$ est une partition de V
Pour i allant de 1 à taille(PP) **faire**
Initialisation de $Best_{cut}$ à ∞ et de $P^{copie} = P^i$
Pour j allant de 1 à Nb_t **faire**
Tirage aléatoire d'un $v_j \in V_{index} \subset P^i$ sans remise
$\{V_{ori}, V_{dest}\}$ = **GGGP_amélioré**(G, 100, v_j)
Si $cut(\{V_{ori}, V_{dest}\}) < Best_{cut}$ ou $cut(\{V_{ori}, V_{dest}\}) < Best_{cut} + 10\%$ **Alors**
Si $cut(\{V_{ori}, V_{dest}\}) < Best_{cut}$ **Alors**
$Best_{cut} = cut(\{V_{ori}, V_{dest}\})$
Suppression des éléments de P^{copie} qui ne respectent pas $< Best_{cut} + 10\%$
Finsi
$V_{index} = V_{ori}$ et ajout de V_{dest} à P^{copie}
Finsi
Fin pour
Ajout de P^{copie} à PP
Fin pour
Suppression des partitions en doublon
Sorties: PP

Cependant, sa manipulation est gourmande en calculs. Contrairement à la première stratégie qui consiste à conserver une unique solution pour chaque bissection, celle-ci consiste à conserver les m bissections qui sont les plus proches de l'optimum. Or, de par la nature itérative hiérarchique de la méthode GGGP, le nombre de bissections à réaliser pour parcourir l'ensemble des solutions se voit considérablement augmenté. Cette stratégie a l'avantage de parcourir plus en profondeur le spectre de solutions du problème de k-partitionnement mais possède le lourd inconvénient de devoir réaliser un nombre de bissections nettement plus important.

Limitation du nombre de bissections

La recherche de l'optimum pour le problème du k-partitionnement nécessite l'exploration de toutes les bissections que peuvent offrir les niveaux de la hiérarchie de partition. En effet, l'application itérative hiérarchique des méthodes de bissection fait que la qualité de la partition 2^{i+1} est dépendante, non seulement de la qualité des bissections réalisées à l'étape précédente, mais aussi de la qualité de la partition 2^i. Partant d'un graphe de taille n, une première salve de bissections est réalisée pour déterminer l'ensemble des solutions au 2-partitionnement. Une salve correspond au fait de tester la méthode GGGP pour chaque sommet de départ. Sur chaque résultat est appliqué une nouvelle salve de bissections (une sur chaque sous ensemble de sommets) pour atteindre le 4-partitionnement. De même, toutes les solutions sont conservées pour faire l'objet de futures bissections pour les niveaux de partitionnement suivants. En somme, afin de parcourir l'intégralité du spectre de solutions du problème de k-partitionnent, il est nécessaire d'appliquer autant de bissections qu'il existe de tirages de sommets de départ pour appliquer la méthode GGGP sur chaque niveau de la hiérarchie de partitions. La nature itérative hiérarchique de la démarche rend l'exploration complète de ce spectre très coûteuse en temps de calculs.

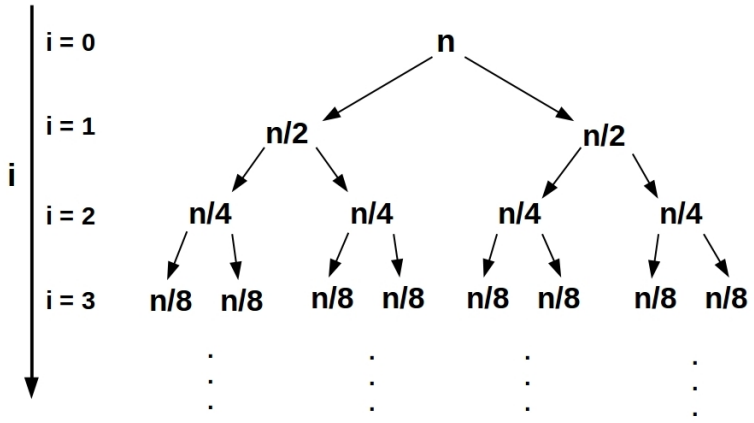


FIGURE 2.11 – Arbre formé par l'application itérative hiérarchique de la méthode GGGP pour un tirage de sommet de départ maximal sur un graphe de taille n

Supposons que le partitionnement soit de la forme $k = 2^l$, pour un graphe de taille n. Partant d'une représentation semblable à celle de la figure 2.11, il est possible d'estimer le nombre maximum de bissections nécessaires pour parcourir la totalité du spectre de solutions du problème de

k-partitionnement contraint à l'aide de la démonstration suivante :

$$2^{l-1} \times n \times \frac{n}{2} \times \frac{n}{4} \times \cdots \times \frac{n}{2^{l-1}}$$
$$2^{l-1} \times \frac{n}{2^0} \times \frac{n}{2^1} \times \frac{n}{2^2} \times \cdots \times \frac{n}{2^{l-1}}$$
$$2^{l-1} \times \prod_{i=0}^{l-1} \frac{n}{2^i}$$
$$2^{l-1} \times \frac{n}{2^{l-1}} \times \prod_{i=0}^{l-2} \frac{n}{2^i}$$
$$n \times n^{l-1} \times \prod_{i=0}^{l-2} \frac{1}{2^i}$$
$$n^l \times \prod_{i=0}^{l-2} \frac{1}{2^i}$$
$$n^l \times \frac{1}{2^K} \text{ avec } K = \sum_{i=0}^{l-2} i = \frac{(l-2) \times (l-1)}{2}$$
$$\frac{n^l}{2^{\frac{(l-2)\times(l-1)}{2}}}$$

En supposant que chaque initialisation de la méthode de bissection GGGP fournisse un résultat différent, et par une application itérative hiérarchique de celle-ci sur chacun des résultats obtenus, il est nécessaire d'effectuer $\frac{n^l}{2^{\frac{(l-2)\times(l-1)}{2}}}$ bissections pour être certain de parcourir le spectre total des solutions qu'offre un graphe de taille n pour un k-partitionnement de la forme $k = 2^l$.

Étudions, à l'aide du tableau 2.3, l'évolution du nombre de bissections nécessaires pour réaliser les partitionnement suivants :

Taille du graphe	$k=2$	$k=4$	$k=8$	$k=16$	$k=32$
200	200	4.10^4	4.10^6	2.10^8	5.10^9
400	400	16.10^4	32.10^6	32.10^8	16.10^{10}
600	600	36.10^4	108.10^6	162.10^8	1215.10^{11}

TABLEAU 2.3 – Tableau référençant le nombre de bissections nécessaires pour réaliser un $k-$partitionnement sur un graphe de taille n

En sachant que, pour notre implémentation, le temps moyen d'une bissection est de 0.0012 seconde, il est possible d'estimer le temps nécessaire pour parcourir l'intégralité du spectre de solutions. Le tableau 2.4 présente ces temps.

Taille du graphe	$k=2$	$k=4$	$k=8$	$k=16$	$k=32$
200	0.24 s	48 s	1.3 h	66.7 h	1666.7 h
400	0.48 s	192 s	10.7 h	1066.7 h	53333.3 h
600	0.72 s	432 s	36 h	5400 h	405.10^5 h

TABLEAU 2.4 – Tableau référençant les temps nécessaires pour réaliser les bissections des différents $k-$partitionnements sur un graphe de taille n

Au vu de ces résultats, il semble naturel de dire qu'il n'est pas envisageable de chercher à parcourir l'intégralité du spectre des solutions. Le coût de calculs qui en résulte, pour un partitionnement supérieur à 4 parties, est beaucoup trop grand. Cependant, en tenant compte des remarques faites en section 2.2.2 sur l'existence de sommets de départ fournissant une même bissection, on peut conclure que ces valeurs fournissent une borne supérieure au nombre de bissections et au temps de calculs nécessaire pour parcourir l'intégralité du spectre des solutions. En réalité, on peut considérer que l'exploration du spectre se fait au prix de $\frac{1}{3}\frac{n^l}{2^{\frac{(l-2)\times(l-1)}{2}}}$ bissections en moyenne, soit une réduction des temps de calculs par 3. Même si cette réduction est non négligeable, elle n'est pas suffisante pour justifier un parcours total du spectre de solutions. C'est pour cela, qu'il a été nécessaire d'employer nos stratégies d'optimisation pour la k-partitionnement.

L'application de notre première stratégie garantit l'obtention d'une partition proche de l'optimum au prix de $l \times n$ bissections. En effet, pour chaque ensemble de sommets de la partition, on teste autant de bissections qu'il y a de sommets dans l'ensemble, mais seule la meilleure solution est conservée. Ainsi, à l'étape suivante, la détermination de la bissection optimale ne nécessite qu'une seule salve de bissections. Pour la seconde stratégie, le nombre de calculs est nettement supérieur car la démarche est plus proche du parcours total du spectre des solutions. En effet, après chaque salve de bissections, l'algorithme conserve les m_i meilleures solutions proche de l'optimum à 10% près. Ainsi l'étape suivante nécessite m_i salves de bissections pour déterminer les m_{i+1} meilleures solutions et ainsi de suite. Le nombre de bissections pour cette stratégie dépend du nombre de solutions conservées à chaque salve de bissections, notons ce nombre m_i en fonction du niveau 2^i. Le nombre de bissections total s'élève à

$$l \sum_{i=0}^{l} m_i \times \frac{n}{2^i}$$

où, $k = 2^l$, n est le nombre de sommets du graphe et $m_0 = n$. La valeur prise par cette formule est difficile à évaluer de par la nature changeante de m_i. Cependant, on peut estimer que pour $k > 4$, sa valeur est très nettement supérieure à celui de la première stratégie.

Même si la première approche ne permet pas d'atteindre certaines solutions fournies par la seconde stratégie, elle reste cependant très fiable en terme de qualité. De plus, le nombre de calculs nécessaires pour déterminer la meilleure solution dans ce cas est nettement plus faible que pour la seconde stratégie. Or, rappelons que cette démarche s'inscrit dans le cadre de l'optimisation des simulations distribuées. Il est donc important que le temps de calculs nécessaire pour le partitionnement soit le plus petit possible, pour ne pas impacter le temps de simulation. L'utilisation de la seconde stratégie semble donc peu indiquée pour satisfaire cette condition de temps. De plus, le gain engendré par l'obtention d'une partition légèrement meilleure ne permettra pas de combler le temps perdu pour la recherche de celui-ci.

Optimisation par affinage local et relaxation de la contrainte de poids

Le choix d'une stratégie garantissant une bonne qualité pour un nombre de calculs limité n'est pas le seul moyen d'optimiser l'application de la méthode GGGP dans un processus ité-

ratif hiérarchique pour le k-partitionnement. En effet, rappelons qu'une partition est considérée comme équilibrée à condition que sa balance soit inférieure ou égale à 1.05. Or, dans la pratique, la méthode **GGGP_améliorée** (12) est conçue pour garantir l'obtention de bissection de balance unitaire ($bal\{V_{ori}, V_{dest}\} = 1$) et donc d'un partitionnement unitaire. En relâchant cette contrainte unitaire à 1.05, il est certainement possible d'améliorer le coût de coupe tout en conservant une partition équilibrée. Cette démarche d'optimisation consiste à appliquer la méthode d'affinage local utilisée par la méthode Multi-niveaux, présentée en section 2.3.2. Celle-ci consiste à déplacer les sommets se trouvant sur le bord d'une partie à condition que son déplacement garantisse une minimisation du coût de coupe et surtout le respect de la contrainte $bal(P_k) \leq 1.05$. Cependant, quand est-il plus judicieux d'appliquer cette stratégie ?

Il existe plusieurs façons d'aborder et d'utiliser cette politique d'affinage pour optimiser le coût de coupe. La première, et la plus forte, consiste à appliquer un affinage sur chaque bissection obtenue par application d'une méthode GGGP. Celle-ci a pour avantage de garantir un coût de coupe minimal mais au prix de très nombreuses applications. Or, rappelons que dans le contexte d'optimisation, une bissection est réalisée pour chaque sommet du graphe choisi comme sommet de départ de la méthode GGGP. Cela implique donc un affinage pour chacune de ces $l * n$ bissections, ce qui peut représenter un obstacle en terme de temps de calculs. De plus, le fait d'affiner chaque bissection ne risque t-il pas de trop contraindre la solution et par conséquent de barrer l'accès à certaines solutions ?

C'est pourquoi nous avons opté pour une politique d'affinage moins restrictive. Celle-ci consiste à affiner chaque partition intermédiaire de la forme 2^i. Ceci a pour avantage de limiter le nombre d'applications de la méthode d'affinage, mais aussi d'améliorer la solution sans trop restreindre le spectre de solutions éligibles. La figure 2.12 schématise l'application de la méthode d'affinage sur la hiérarchie de partitions.

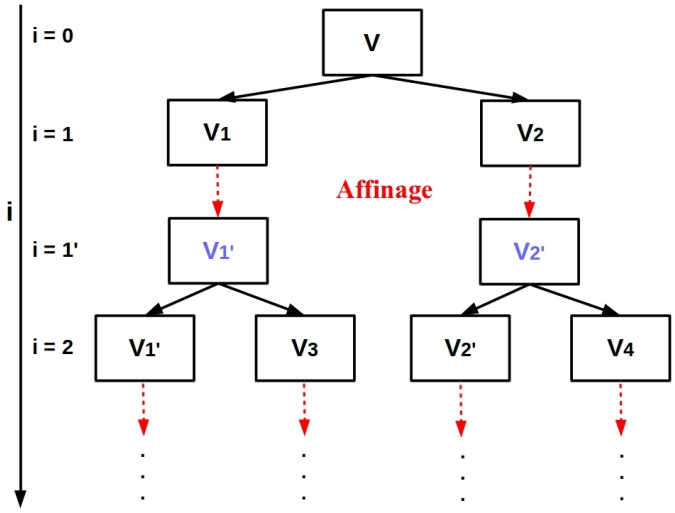


FIGURE 2.12 – Application d'une méthode d'affinage locale sur chaque niveau de la hiérarchie de partitions

Le fait d'affiner exclusivement la partition lorsqu'elle est de la forme 2^i garantit un équilibre de charge entre les parties. Par exemple, pour $k = 2^1$ $P_2 = \{V_1, V_2\}$, où V_1 et V_2 sont parfaitement équilibrées ce qui garantit une balance unitaire. Or, si on décide d'appliquer un affinage avant d'at-

teindre une partition de la forme 2^i, celle-ci peut par exemple se composer de trois sous-ensembles $\{V_1, V_2, V_3\}$, où V_1 et V_3 ont un poids égal à la moitié de celui de V_2. Une telle partition n'étant pas équilibrée, il est impossible de trouver des sommets dont le déplacement satisfasse la condition $bal(P_k) \leq 1.05$. L'inclusion de cette optimisation se fait au sein de la méthode de bissection itérative hiérarchique, présentée en section 1.4.4. Celle-ci donne naissance à l'algorithme 17 présenté ci-dessous :

Algorithme 17 Algorithme de bissection itérative hiérarchique avec affinage des 2^i-partitions

PROCÉDURE: Bissection_Itérative_Hiérarchique_2i_affinage
ENTRÉES: $G = (V, E)$, k le nombre de parties
  Calcul de la profondeur $i_{max} = \frac{\ln(k)}{\ln(2)}$
  Initialisation de la partition $P = V_1$, où $V_1 = V$
  Pour i allant de 0 à i_{max} **faire**
    Pour j allant de 1 à 2^i **faire**
      Bissection de l'ensemble V_j
      Ajout de V_{j+i} à P
      Si Modification du graphe nécessaire **Alors**
        Mise à jour de G par suppression des arcs reliant les $V_k \subset P$
      Finsi
    Fin pour
    $P =$ Affinage(P)
  Fin pour
  Sorties: P

L'utilisation de cette dernière pose cependant son lot de problèmes. En effet, la relaxation de la contrainte d'équilibre sur les niveaux de la hiérarchie de partitions peut être à l'origine du non respect de celle-ci pour un partitionnement de grande taille. Prenons par exemple un graphe de taille 200 dont le 8-partitionnement incluant l'affinage est illustré par la figure 2.13. La première bissection donne une 2-partition équilibrée, éligible à un affinage. Après celui-ci, le coût de coupe est minimisé et la balance vaut 1.05. Pour atteindre la 4-partition, une bissection est appliquée sur chaque sous-ensemble de la 2-partition. Or, dans ce cas, la partition n'est pas équilibrée car elle possède une balance de 1.06. Il est donc impossible de réaliser un affinage. On constate le même procédé sur chacun des niveaux suivants de la hiérarchie de la partition. De plus, le déséquilibre provoqué au niveau $i = 1$ s'amplifie au fil des niveaux pour atteindre une balance de 1.08 pour la 8-partition.

L'application systématique d'un affinage pour respecter la contrainte $bal(P_k) \leq 1.05$ engendre un déséquilibre à partir d'un certain niveau de la hiérarchie de partition. Cependant, il existe deux moyens de contrôler cette dérive :

— contraindre l'équilibre de la partition par application d'une méthode d'équilibre de charge. Même si cette approche est en contradiction avec la notion d'affinage, elle permet de contourner le problème d'équilibre et d'appliquer par la suite un nouvel affinage.

— contrôler le paramétrage de la contrainte d'équilibre (la balance maximum autorisée) en fonction du niveau de la hiérarchie i et de la taille de la partition $k = 2^l$. Par exemple, pour un 8-partitionnement la balance maximum autorisée du niveau $i = 1$ ne doit pas être supérieure

à 1. De façon générale, la balance maximum autorisée évolue en fonction du niveau de la partition sur laquelle elle se situe. Partant d'une balance de 1 par une bi-partition, elle augmente de $\frac{0.05}{l}$ par niveau.

Cette seconde utilisation de l'affinage sur la hiérarchie de partition garantit une évolution homogène des parties de la partition, de façon à ne pas aller à l'encontre de l'équilibre de la partition.

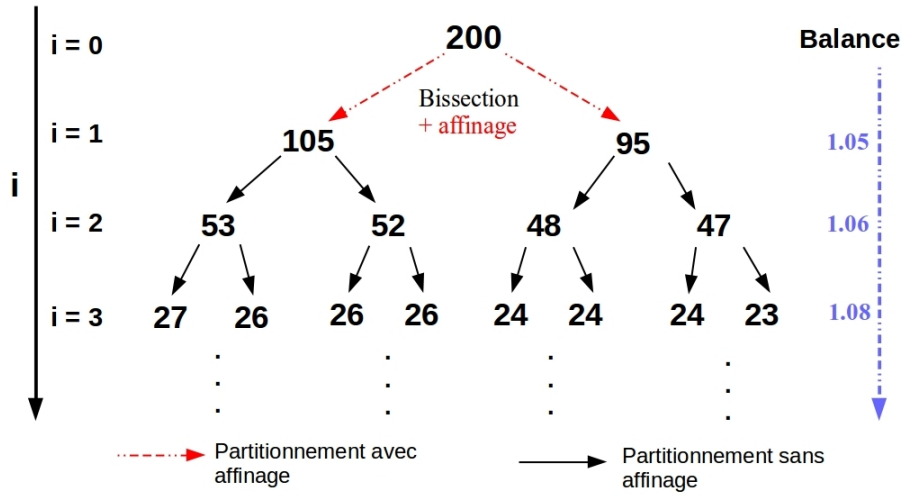


FIGURE 2.13 – Évolution de l'équilibre de la partition au cours des étapes du partitionnement par application d'une méthode d'affinage

La qualité de la partition n'est pas exclusivement dépendante de la phase de partitionnement. Rappelons que les simulations de grandes tailles donnent naissance à des graphes de grandes tailles. Il est donc nécessaire d'inclure notre méthode de partitionnement dans un algorithme multi-niveaux pour garantir son efficacité et sa rapidité. Or, celui-ci nécessite également un paramétrage et une optimisation de ces différentes phases pour garantir une partition optimale. La section suivante présente les améliorations apportées à ces différentes phases.

2.3 Optimisation des phases de la méthode multi-niveaux

Lors de la manipulation de graphes de grande taille, l'optimisation des méthodes de partitionnement n'est pas le seul gage d'une partition de qualité. En effet, dans ce contexte, les méthodes de partitionnement sont généralement employées de manière sous-jacente dans une méthode Multi-niveaux. Or, la qualité globale de la partition repose entièrement sur l'optimalité de chacune de ces phases. Cette sous-section présente les améliorations apportées aux différentes phases de la méthode Multi-niveaux.

2.3.1 Optimisation de la phase de contraction

L'objectif de la phase de contraction est de fournir une base de graphes réduits dont le poids des arcs doit être le plus faible possible. En effet, cette contrainte permet d'anticiper une future minimisation du coût de coupe par application d'une méthode de partitionnement de graphes. Les

principaux algorithmes de contraction présents dans la littérature, et présentés en section 1.5.2, reposent sur la notion d'agrégation maximale qui consiste à "fusionner" les paires de sommets reliées par un arc de poids fort. Chaque méthode utilise un procédé différent pour parvenir à cette agrégation maximale. Au cours de cette thèse, nous avons étudié trois méthodes de contraction : la méthode d'Hendrickson-Leland (10), la méthode HEM (11) et la méthode SHEM. L'étude des points forts et des faiblesses de ces méthodes nous a permis de créer notre propre méthode de contraction.

Faiblesses des méthodes existantes

Les méthodes d'Hendrickson-Leland et HEM reposent sur un même mécanisme de tirage aléatoire de sommets mais se différencient dans la sélection du sommet voisin à "fusionner". Même si la méthode HEM offre une contraction de meilleure qualité, l'aspect aléatoire des tirages de sommets ne garantit pas l'obtention d'une même contraction pour deux appels de la méthode sur un même graphe. La figure 2.14, montre deux contractions d'un même graphe par application de la méthode HEM. Cet aspect aléatoire de la contraction est un frein à l'anticipation d'un partitionnement de qualité.

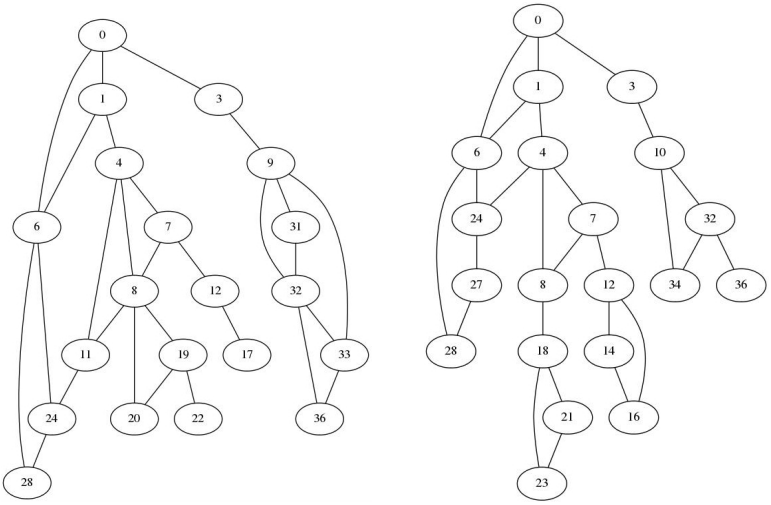


FIGURE 2.14 – Exemples de contraction d'un même graphe suivant la méthode HEM pour des tirages différents

Pour remédier à cela et renforcer la contrainte de fusion de sommets de poids forts, les auteurs de la méthodes HEM propose une amélioration de celle-ci : la méthode SHEM. Comme présenté en section 1.5.2, cette méthode remplace le tirage aléatoire de sommets par un tirage suivant l'ordre décroissant de degré des sommets du graphe. Ainsi, cette amélioration garantit l'obtention systématique d'un même graphe contracté. De plus, ce système de tirage est supposé garantir la fusion de sommets reliés par un arc de poids fort. Or, dans la pratique cette condition n'est pas toujours garantie. Prenons l'exemple de la figure 2.15, un sommet de degré important ne possède pas forcément un arc de poids fort. En effet, prenons l'exemple du couple de sommets de gauche, le sommet v_1 possède 8 voisins dont les arcs qui les lient ont un poids inférieur à 0.45 et admet un degré de 1.85. Pour le couple de sommets de droite, le sommet v_3 possède 2 voisins dont les arcs

qui les lient ont un poids inférieur à 1.3 et admet un degré de 1.55. Dans le contexte d'utilisation de la méthode SHEM, le sommet v_1 est sélectionné en premier car il admet un plus grand poids que v_3. Cependant, ce choix s'avère ne pas être judicieux car l'arc de poids maximum sortant de v_1 n'est que de 0.45 contre 1.3 pour celui de v_3. Pour être efficace, la méthode de contraction doit favoriser la fusion des sommets v_3 v_4 à celle de v_1 v_2.

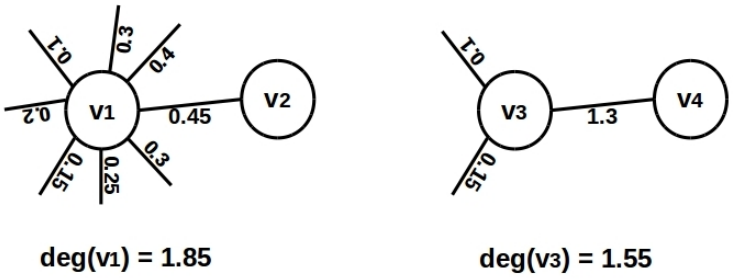


FIGURE 2.15 – Mise en évidence des lacunes de la sélection de sommets par degré maximum de la méthode SHEM

Pour corriger ces "problèmes", nous proposons une méthode de contraction reposant sur la sélection des arcs et non des sommets. L'objectif de celle-ci est identique aux méthodes de la littérature, mais permet de supprimer l'aspect aléatoire du résultat tout en garantissant la minimisation des arcs de poids forts.

Mise en place de l'algorithme

Afin de garantir une contraction dont le poids des arcs qui la compose soit minimal, il est nécessaire de forcer la méthode de contraction à fusionner les sommets reliés par des arcs de poids forts. Même si la littérature offre des méthodes permettant de s'en approcher, elle ne garantit nullement la convergence vers un graphe contracté idéal. Notre approche consiste à manipuler directement les arcs du graphe pour garantir un appariement maximal. Elle peut être considérée comme une amélioration de la méthode de contraction d'Hendrickson-Leland, car elle repose sur le même principe de sélection d'arcs mais sans l'aspect aléatoire.

Cette méthode de contraction commence par trier les arcs du graphe par ordre décroissant de poids. Ainsi, ce classement permet de garantir la fusion des sommets reliés par un arc de poids fort. Pour éviter la fusion de sommets multiples, il est nécessaire de verrouiller ceux ayant déjà subi une fusion. L'ensemble V_{verrou}, initialement vide, contient l'ensemble des sommets non éligibles à une fusion. Pour chaque arc sélectionné, l'algorithme récupère les sommets se trouvant à chaque extrémité (v_i et v_j) et vérifie s'il l'un des deux se trouve dans V_{verrou}. Si ce n'est pas le cas, les deux sommets sont ajoutés à V_{verrou} et la fusion a lieu. Au cours de cette fusion, le sommet ayant l'indice le plus faible est conservé. L'arc reliant v_i à v_j est détruit et tous les arcs adjacents à v_j sont rattachés à v_i à condition qu'ils n'existent pas déjà. Si tel est le cas, seule la fusion des poids a lieu. Le processus s'arrête lorsque tous les arcs ont été sélectionnés ou que tous les sommets ont été verrouillés. Cette méthode est utilisée au sein d'une structure itérative, communément nommée *"passe"*, pour garantir l'obtention d'un graphe contracté de la taille souhaitée. Le fonctionnement

de notre méthode de contraction est présenté par l'algorithme 18.

Algorithme 18 Algorithme de contraction de sommets reposant sur une amélioration de la méthode d'Hendrickson-Leland

PROCÉDURE: Contraction_HL_améliorée
ENTRÉES: $G = (V,E)$, $taille_{max}$ taille souhaitée pour la contraction
Création d'un base de graphe $Base_G$
Ajout de $G = G_1$ à la base de graphe $Base_G$
$i = 1$
Tantque $taille(G_i) > taille_{max}$ *("passe")* **faire**
G'_i = Copie(G_i)
$V_{verrou} = \emptyset$
Trie de E'_i par ordre décroissant de poids
Tantque des arêtes sont disponibles dans E'_i ou que $V_{verrour} \neq V'_i$ **faire**
Tirage de la première arête $e \in E'_i$ non verrouillée tel que $e = (v_1, v_2)$
Suppression de e dans E'_i
Si $v_1 \notin V_{verrou}$ et $v_2 \notin V_{verrou}$ **Alors**
Fusion des poids : $poids(v_1) = poids(v_1) + poids(v_2)$
Pour chaque arrête adjacente à v_2 (e_2) **faire**
Si si elle possède une extrémité commune avec une des arêtes adjacentes à v_1 (e_1)
Alors
Fusion des poids : $poids(e_1) = poids(e_1) + poids(e_2)$
Suppression de e_2
Sinon
Création d'une nouvelle arête reliant v_1 à l'extrémité de e_2
Suppression de e_2
Finsi
Fin pour
Verrouillage des arêtes adjacentes à v_1
Suppression du sommet v_2 dans V'_i
Ajout de v_1 et v_2 à V_{verrou}
Finsi
Fin Tantque
$G_{i+1} = G'_i$
Ajout de G_{i+1} à la base de graphe $Base_G$
$i = i + 1$
Fin Tantque
Sorties: $Base_G = \{G_1, ..., G_n\}$

Exemple de contraction

Cette sous-section présente un exemple de contraction de graphe de petite taille dans le but d'illustrer parfaitement le fonctionnement de notre méthode de contraction. Partant du graphe de taille 11 présenté à gauche dans la figure 2.16, où chaque sommet possède un poids unitaire, on applique la méthode de contraction d'Hendrickson-Leland améliorée. Celle-ci commence par trier les arcs par ordre décroissant de leur poids. Dans cet exemple, quatre arcs admettent le poids maximum de 0.1546. L'algorithme choisit celui qui a été classé premier par l'algorithme de tri

(classement arbitraire dans ce cas), l'arc reliant les sommets 9 et 10. Les sommets 9 et 10 n'étant pas verrouillés, la fusion a lieu. Le sommet 9 est conservé, tandis que le sommet 10 est détruit ainsi que l'arc les reliant. Avant sa destruction, on incrémente le poids du sommet 9 par celui du sommet 10 et on rattache tous les arcs sortant de 10 au sommet 9 (sauf l'arc reliant 9 et 10). Les arcs adjacents au sommet 9 sont verrouillés pour éviter une nouvelle sélection du sommet. Une fois la fusion terminée, le processus de sélection reprend tant que la taille souhaitée n'est pas atteinte. Au cours de celui-ci, les couples de sommets (7,8), (5,6), (1,4) et (2,3) sont fusionnés dans cet ordre. L'algorithme s'arrête suite à cette dernière fusion. Le graphe de droite de la figure 2.16 présente le résultat de la contraction du graphe de taille 11.

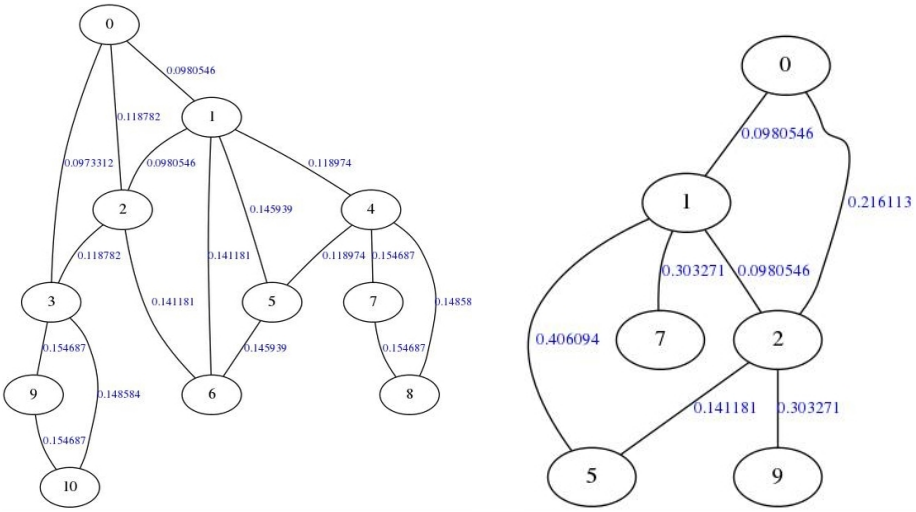


FIGURE 2.16 – Exemple de contraction d'un graphe de taille 11 par application de la méthode d'Hendrickson-Leland améliorée

Une information est toute fois manquante sur cette représentation de graphes, lors de la contraction le poids des sommets fusionnés change. Ici, après contraction, les sommets 1, 2, 5, 7, 9 ont un poids de 2. Ce poids a une grande importance pour la phase de partitionnement, il est donc important de bien le prendre en compte.

Nous avons vu en section 1.4.4 que certaines méthodes de partitionnement, comme la GGGP, ne s'appliquent que sur des graphes de taille réduite. Le niveau de contraction dépendant en partie du niveau de finesse souhaité mais aussi de la vitesse de convergence de l'algorithme. Cependant, en fonction du nombre de parties souhaitées, il n'est pas toujours judicieux, voire possible, de contracter le graphe au niveau souhaité.

Paramétrage du niveau de contraction en fonction du nombre de partitions

Dans l'absolu, pour appliquer efficacement une méthode GGGP, il serait souhaitable de contracter le graphe jusqu'à atteindre une taille inférieure à 200 sommets. Cependant, dans la pratique, ce souhait n'est pas toujours envisageable. En supposant que le poids des sommets est unitaire, le partitionnement d'un graphe de taille 200 en k parties donne naissance à des sous-ensembles de sommets dont la taille est d'environ $\frac{200}{k}$. Pour k allant de 2 à 128, le nombre de sommets moyen par partie évolue comme suit :

Nombre de parties **k**	2	4	8	16	32	64	128
Nombre moyen de sommets	100	50	25	12	6	3 à 4	1 à 2

Au delà de $k = 8$, le nombre moyen de sommets par partie devient trop faible ce qui peut avoir pour effet de réduire le spectre des solutions. Pour remédier à cela, nous proposons de garantir un nombre minimum de 15 sommets par partie (pour une pondération unitaire des sommets). Pour respecter cette contrainte, il est nécessaire de modifier le niveau de contraction pour qu'il soit en accord avec celle-ci. Le niveau de contraction devient un paramètre variable, dépendant du nombre de parties k :

$$\begin{cases} taille_{max} = 600 \textbf{ si } k \times 15 > 600 \\ taille_{max} = k \times 15 \textbf{ si } k \times 15 \leq 600 \\ taille_{max} = 200 \textbf{ si } k \times 15 < 200 \end{cases}$$

La relaxe du niveau de contraction se limite cependant à 600 pour éviter de perdre en qualité de partitionnement. Il est possible de modifier cette nouvelle contrainte dans le cadre de l'utilisation d'une méthode spectrale par exemple.

2.3.2 Création d'une méthode d'affinage local

La littérature offre des approches permettant d'affiner localement des bissections de graphes. C'est notamment le cas de l'algorithme d'affinage de Kernighan-Lin, présenté en section 1.5.3. Or, dans notre cas, nous cherchons à réaliser un affinage direct de la partition sans être obligé d'affiner chaque bissection deux à deux. De plus, cette méthode est initialement conçue pour affiner des partitions de sommets non pondérés, où les parties sont de même taille. Dans notre cas, les graphes sont tous valués et ne sont pas contraints en terme de taille mais seulement en terme de poids. C'est pourquoi, nous avons créé une méthode d'affinage, fortement inspirée de celle de Kernighan-Lin, permettant une généralisation au k-partitionnement. Le principe de l'algorithme est similaire au leur, tout sommet ayant une différence de coupe (voir 1.5.3) positive est éligible à un déplacement vers une partie voisine. Cependant, le nombre multiple de parties a nécessité une légère modification de cette notion de différence de coupe.

Généralisation de la différence de coût de Kernighan-Lin au k-partitionnement

Comme présenté en section 1.5.3, la différence de coupe $Diff(v)$ d'un sommet $v \in V_1$, correspond à la différence entre son coût extérieur $Ext(v)$ et son coût intérieur $Int(v)$. Or, pour une bissection de graphe $P_2 = \{V1, V2\}$, si le sommet v' adjacent à v est :

— contenu dans $V1$, alors le poids de l'arc (v, v') incrémente la valeur de $Int(v)$

— contenu dans $V2$, alors le poids de l'arc (v, v') incrémente la valeur de $Ext(v)$.

Ce qui est plus complexe pour le k-partitionnement.

Prenons l'exemple de la figure 2.17, pour le sommet **v** le coût intérieur équivaut à la somme des poids des arcs reliant **v** aux sommets adjacents appartenant à la même partie, ici $Int(v) = 5.4$. Cette notion de coût intérieur n'est pas sensible au k-partitionnement, sa définition reste la même.

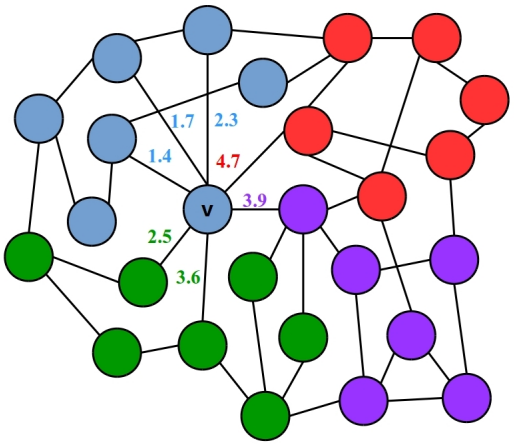


FIGURE 2.17 – Adaptation de la notion de différence de coupe au k-partitionnement

Cependant, pour le coût extérieur, le sommet **v** possède des voisins de différentes parties de la partition (rouge, vert, violet). Pour savoir si le déplacement de **v** vers l'une des parties voisines permet d'engendrer un gain, il est nécessaire de calculer le coût extérieur pour chacune de ces parties. L'adaptation au k-partitionnement nécessite la modification du coût extérieur $Ext(v)$ en un ensemble de coût extérieur pour chaque partie voisine à v. Notons ce nouveau coût extérieur $Ext_k(v) = \{Ext_i(v)\}\ \forall i \in [1,k]$. Ici, $Ext_k(v) = \{4.7, 6.1, 3.9\}$. La différence de coupe subit quant à elle aussi une légère modification pour le k-partitionnement. Lors de l'affinage, on cherche à déplacer un sommet vers une partie qui maximise le gain (et qui minimise donc le coût de coupe). Il est donc important de choisir la partie voisine dont le Ext_i est maximum. La nouvelle définition de la différence de coupe pour le k partitionnement est donc :

$$Diff_k(v) = \max_i Ext_i(v) - Int(v)$$

Dans cet exemple, la partie voisine dont le coût extérieur est le plus grand est la partie verte avec un $Ext_i(v) = 6.1$. C'est donc cette partie qui peut être éligible pour recevoir le sommet **v**. La différence de coupe de ce sommet est de $Diff_k(v) = 6.1 - 5.4 = 0.7 > 0$, donc **v** sera déplacé de la partie bleue à la partie verte à condition que la balance de partitionnement soit conservée.

En utilisant l'ancienne version de la différence de coupe, le coût extérieur équivaut à $Ext(v) = \sum_{i=1}^{k} Ext_i$. Or dans ce cas, $Ext(v) - Int(v)$ peut être supérieur à 0, alors qu'en réalité $Diff_k(v)$ ne l'est pas. Prenons l'exemple de la figure 2.18, $Ext_k(v) = \{1.2, 1.3, 1.6\}$, $Ext(v) = 4.1$ et $Int(v) = 3.7$. En calculant l'ancienne différence de coupe on a $Diff(v) = 4.1 - 3.7 = 0.4 > 0$, ce qui autorise l'algorithme à déplacer le sommet **v** vers l'une de ces parties voisines, alors que $Diff_k(v) = 1.6 - 3.7 = -2.1 < 0$. Cet exemple souligne l'importance d'adapter la notion de différence de coupe au k-partitionnement pour éviter ce genre de dérive. En effet, une telle erreur peut forcer l'algorithme d'affinage à effectuer un déplacement préjudiciable pour le coût de coupe de la partition.

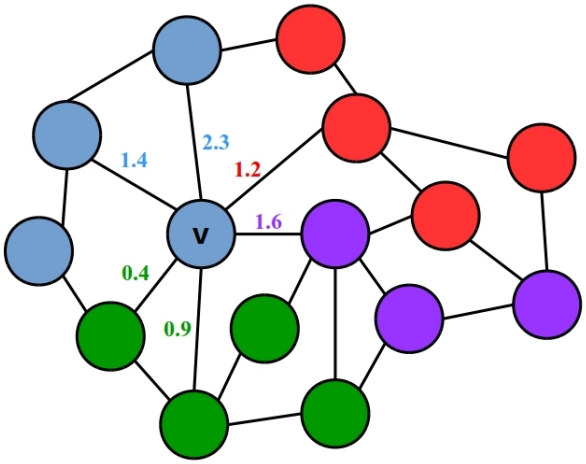


FIGURE 2.18 – Illustration de la différence entre $Ext(v)$ et $Ext_k(v)$

Mise en place de l'algorithme

L'affinage repose exclusivement sur la différence de coupe présentée ci-dessus. En effet, tout sommet ne se situant pas sur le bord d'une partie aura forcément une différence de coupe négative. Ce qui permet de déterminer un sous-ensemble de sommets éligibles à un déplacement au prix de quelques opérations basiques. La première étape de l'algorithme d'affinage consiste à calculer la différence de coupe de chaque sommet et de ne conserver que les sommets dont la différence de coupe est positive. Ces sommets sont conservés dans un ensemble nommé V_{dep} et sont triés par ordre décroissant de leur différence de coupe $D_1 \geq D_2 \geq \cdots \geq D_n$. Ainsi, le sommet de plus fort gain (D_1) effectuera son déplacement en premier. Rappelons tout de même que le déplacement n'a lieu que si la balance de partitionnement respecte la condition $bal(P_k) \leq 1.05$ après le déplacement.

Après chaque déplacement, l'ensemble V_{dep} est mis à jour en recalculant la différence de coupe de chaque sommet présent dans V_{dep}, à l'exception du sommet précédemment déplacé (D_1) qui est supprimé. On calcule ensuite la différence de coupe des sommets adjacents à celui de D_1 et on les ajoute à V_{dep} si leur différence de coupe est positive. Le processus de déplacement est réitéré jusqu'à ce que plus aucun sommet ne soit éligible à un déplacement (V_{dep} vide). L'algorithme 19 présente le fonctionnement de notre méthode d'affinage local.

Il est possible de durcir la contrainte de poids des parties en ajoutant le paramètre bal_{max} au niveau de la condition (*) de l'algorithme. En effet, comme nous l'avons vu en section 2.2.3, l'utilisation par itération hiérarchique d'une méthode GGGP peut nécessiter la modulation de cette contrainte lors de l'application de l'affinage.

Cette méthode a le double avantage d'être parfaitement autonome et ne nécessite que peu de calculs. En effet, une fois la différence de coupe calculée pour chaque sommet à l'initialisation, seuls les sommets éligibles à un déplacement sont recalculés au cours de l'exécution de l'algorithme. La minimisation du nombre de calculs est véritablement importante, car l'algorithme d'affinage est appelé autant de fois qu'il y a de graphes dans la base de graphe contracté lorsqu'il est utilisé dans la méthode Multi-niveaux. Or, ces graphes ont une taille allant de 200 à plusieurs centaines de milliers de sommets. La sous section qui suit illustre le fonctionnement de cette méthode

d'affinage à l'aide d'un exemple.

Algorithme 19 Algorithme d'affinage local reposant sur la différence de coupe adaptée au k-partitionnement

PROCÉDURE: affinage_local
ENTRÉES: $G = (V,E)$, P_k
Calcule de chaque $Diff_k(v)$ $\forall v \in V$
Chaque v dont $Diff_k(v) > 0$ est ajouté à V_{dep}
Trie de V_{dep} par ordre décroissant des $Diff_k(v)$
Tantque $taille(V_{dest}) \neq 0$ **faire**
Tirage du v_{max} de différence de coupe maximum
Si le déplacement de v_{max} vers la partie voisine respecte $bal(P_k) \leq 1.05$ (*) **Alors**
Déplacement de v_{max} vers la partie voisine
Recherche des sommets adjacents à v_{max}
Pour chaque sommet v' adjacent à v_{max} **faire**
Calcule du $Diff_k(v')$
Si $Diff_k(v') > 0$ **Alors**
Ajout de v' à V_{dest}
Finsi
Fin pour
Recalcule des $Diff_k(v_i)$ pour chaque $v_i \neq v_{max}$
Finsi
Suppression de v_{max} dans V_{dest}
Trie de V_{dep} par ordre décroissant des $Diff_k(v)$
Fin Tantque
Sorties: P_k

Exemple d'utilisation de l'algorithme d'affinage local

La figure 2.19 illustre l'affinage d'une 4-partition par application de notre méthode d'affinage locale. Dans le cadre de cet exemple, nous ne tiendrons pas compte de la balance de partitionnement car pour qu'un graphe de petite taille soit équilibré, il faut que le poids de chaque partie soit identique. Il est donc impossible d'effectuer le moindre déplacement sans porter préjudice à la balance. Le graphe en haut à gauche représente la partition avant l'application d'un affinage. Celle-ci admet un coût de coupe de 21.7 (par soucis de clarté, tous les poids ne sont pas représentés). Dans cette partition, trois sommets sont éligibles à un déplacement :

- v_1 a pour différence de coupe $Diff_k(v_1) = 8.1 - 5.4 = 2.7 > 0$, il peut être déplacé vers la partie verte

- v_2 a pour différence de coupe $Diff_k(v_2) = 4.4 - 3.4 = 2 > 0$, il peut être déplacé vers la partie violette

- v_3 a pour différence de coupe $Diff_k(v_3) = 3.4 - 2 = 1.4 > 0$, , il peut être déplacé vers la partie violette

Les sommets sont triés suivant la valeur de leur différence de coupe. C'est donc le sommet v_1 qui réalise en premier son déplacement. Les graphes du haut illustrent le déplacement du sommet

v_1 de la partie bleue à la partie verte. Ce déplacement réduit le coût de coupe de 2.7, le faisant passer de 21.7 à 19. La différence de coupe est recalculée pour chaque sommet précédemment retenu et pour ceux adjacents à v_1. C'est ensuite le sommet v_2 qui est éligible à un déplacement. Le déplacement de celui-ci vers la partie violette engendre un gain de coupe de 2, faisant passer le coût de coupe de 19 à 17. Le processus est réitéré et seul le sommet v_3 est éligible à un déplacement vers la partie violette. Ce déplacement réduit le coût de coupe de 1.4 le faisant passer de 17 à 15.6.

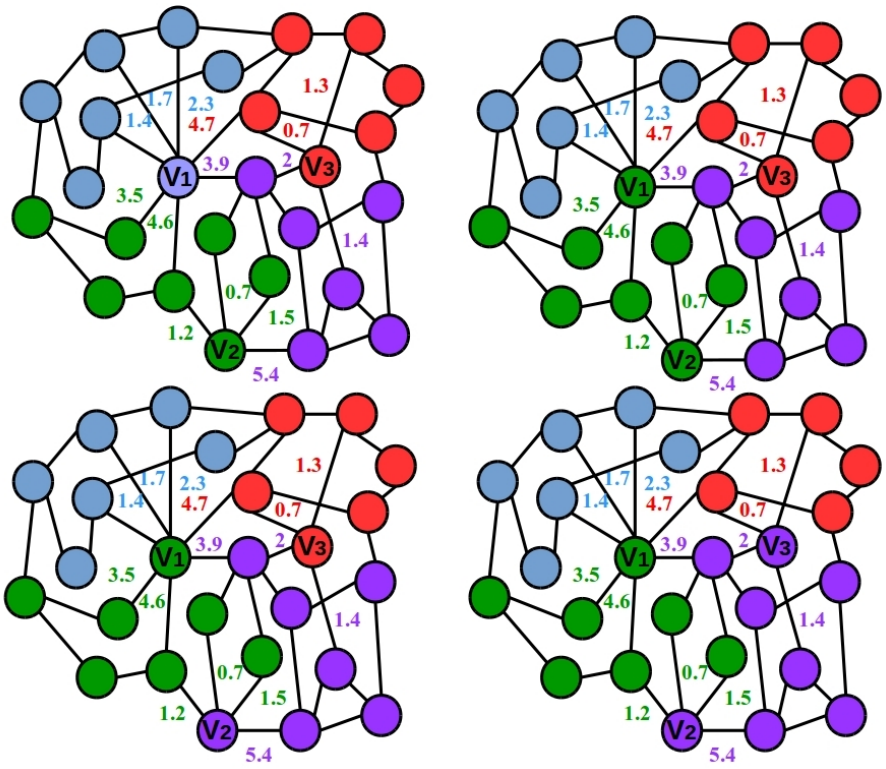


FIGURE 2.19 – Exemple d'évolution d'une partition par affinages successifs

Sur cet exemple fictif, l'application de la méthode d'affinage permet une réduction du coût de coupe de 21 à 15.6. Dans la réalité, la relâche de la contrainte d'équilibre de la partition permet de réduire le coût de coupe à hauteur de 20 à 25%.

Conclusion

La réduction du temps des simulations distribuées DEVS repose sur une distribution efficace des modèles parmi l'ensemble des noeuds de calcul disponibles. Hormis le modélisateur qui a une parfaite connaissance de ces modèles, il est impossible de déterminer une distribution homogène sans étudier le réseau de connexions qui les unissent. Rappelons qu'il existe deux structurations possibles par un même modèle :

— une structure hiérarchique qui se compose d'une hiérarchie de modèles couplés et de modèles atomiques

— une structure à plat qui se compose d'un unique modèle couplé contenant tous les modèles atomiques

Cette dernière permet d'obtenir le graphe de modèles fournissant l'information nécessaire à la distribution homogène des modèles. En effet, ce graphe permet de connaître le lien de dépendance entre les modèles et, en théorie, de connaître la quantité d'informations en transit entre eux.

À partir de ce graphe, il est possible de créer une partition de modèles de façon à respecter un équilibre de charge et une minimisation de messages en transit entre les noeuds de calcul. L'optimalité de la distribution repose entièrement sur ce partitionnement, c'est pourquoi il est nécessaire d'optimiser au mieux chacune de ces étapes. Le rôle de ce chapitre, outre la présentation de la création d'une structure DEVS optimale, est de montrer l'importance de l'attention portée aux méthodes de partitionnement pour garantir une partition de qualité et par conséquent une distribution optimale.

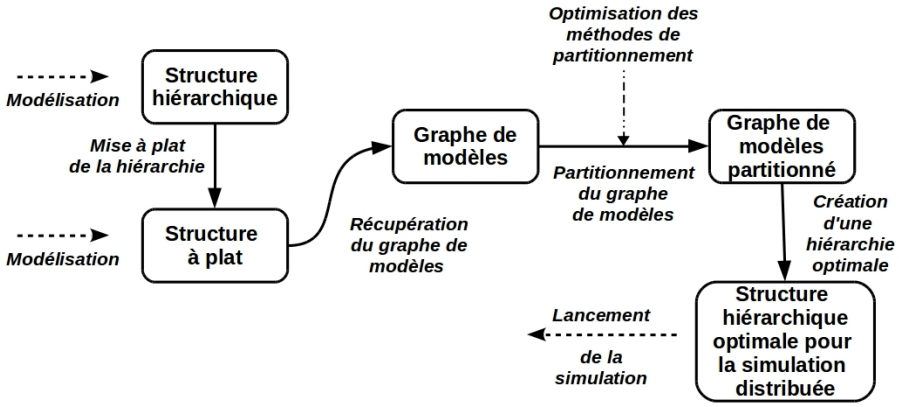


FIGURE 2.20 – Schéma bilan de la procédure d'optimisation des simulations distribuées DEVS

La figure 2.20 récapitule chacune des étapes nécessaires pour parvenir à une distribution optimale d'un modèle DEVS. Cependant, le graphe de modèle fourni par la mise à plat de la hiérarchie ne contient pas l'information sur la quantité de messages en transit entre chaque modèle mais seulement le lien de dépendance qui les unissent. Par manque d'information, le poids des arcs était initialement fixé à 1. Or, dans [HQR15b] nous nous sommes aperçus que ce manque d'information impacte fortement la qualité de la distribution des modèles et par conséquent le temps de simulation. En effet, émettre cette information revient à ne pas prendre en compte la dynamique des modèles DEVS. Or, la dynamique de la simulation repose exclusivement sur chacune des dynamiques individuelles des modèles atomiques. La réponse à ce problème est d'apprendre la dynamique individuelle des modèles afin de déterminer la véritable pondération du graphe. C'est l'objet du prochain chapitre.

Chapitre 3

Apprentissage individuel de la dynamique des modèles atomiques pour une pondération efficace du graphe

L'apprentissage d'un modèle est l'art de prédire une donnée de sortie y à partir d'une donnée d'entrée x, qui dans notre cas correspond plutôt à une séquence d'observations du modèle en question. De manière plus générale, l'apprentissage revient à produire la meilleure décision δ à partir d'une donnée d'entrée x qui permettrait de générer une donnée y qui n'est pas connue au moment où cette décision est prise. Cet apprentissage peut être utilisé à diverses fins, comme par exemple pour faire de la prédiction, de la classification de données et dans le but de créer un automate à états probabilistes générant des séquences d'observations statistiquement proche de celles de la dynamique d'un modèle appris. C'est pour cette dernière utilisation que nous avons opté dans nos articles [HQR15c] et [HQR16]. En effet, la pondération du graphe de modèles repose sur la connaissance et l'estimation de la quantité de données en transit entre chaque modèle atomique. Cette estimation passe par la connaissance de la dynamique qui anime chaque modèle et nécessite par conséquent un apprentissage de celle-ci. Mais quelle méthode d'apprentissage utiliser ? Et dans le but de créer quel type d'automate ? Ces questions trouvent réponses en section 3.2.

Les outils nécessaires à la mise en place d'un apprentissage ne sont pas nouveaux, cependant leur utilisation dans un contexte DEVS l'est. En effet, les particularités liées à ce formalisme nécessitent certaines modifications des outils d'apprentissage, tant sur le plan théorique que pratique. La génération de séquences d'observation, utilisée comme base d'apprentissage, nécessite la simulation individuelle de chaque dynamique des modèles atomiques qui compose le modèle DEVS global. Or cette simulation est elle-même dépendante d'éléments externes. Cette section a pour objectif de présenter le type d'automate utilisé pour simuler la dynamique des modèles. Pour cela, il a été nécessaire de réaliser une étude des outils existants dans le but de trouver celui qui offre la meilleure solution en vue des éléments disponibles. Cette section présente également les modifications apportées à la méthode d'apprentissage de notre choix, issue de la littérature, pour prendre en compte les particularités d'un modèle DEVS. Une validation de notre approche ainsi

qu'une phase de tests sont réalisées pour estimer la qualité de notre processus d'apprentissage.

3.1 Vers une pondération autonome du graphe de modèles

L'optimalité de la distribution des modèles repose en grande partie sur la qualité du partitionnement du graphe de modèles. Or, ce partitionnement est réalisé à partir des informations contenues dans le graphe qui proviennent elles-même de la modélisation. En effet, il existe deux catégories d'informations :

— les explicites qui sont liées à la structuration du modèle. Elles sont obtenues lors de la mise à plat de la hiérarchie et sont, par exemple, la forme et la taille du graphe. Ces informations s'obtiennent par une simple observation de la structure du modèle.

— les implicites qui sont liées à la dynamique du modèle. Chaque modèle atomique qui compose le modèle peut posséder une dynamique différente. Ces dynamiques influent sur les interactions entre modèle atomique, comme par exemple la fréquence ou le poids des messages émis. Ces informations sont contenues dans la pondération du graphe de modèles, mais ne sont pas directement fournies par une simple étude de la structure du modèle.

La structure à plat du modèle DEVS fournit toutes les informations explicites nécessaires à la création du graphe de modèles. Cependant, une simple étude de celle-ci ne suffit pas à fournir toutes les informations nécessaires au bon déroulement d'un partitionnement. En effet, le graphe de modèles ainsi créé ne contient aucune information implicite. Ce manque d'information nous a contraint, dans un premier temps, à mettre une pondération unitaire sur l'ensemble du graphe. Il est bien évident que ce n'est pas suffisant.

Partitionner un graphe où la pondération ne reflète en rien la dynamique du modèle simulé revient à distribuer "au hasard" les modèles atomiques sur les clusters. Cette notion de "hasard" reste tout de même exagérée, car l'utilisation de nos méthodes garantit une certaine logique spatiale dans la distribution des modèles atomiques. Cependant, la partition de coût de coupe minimum obtenue à partir du graphe de modèles à pondération unitaire a de très fortes chances de ne pas être la solution optimale pour le graphe de modèles pondéré par les informations implicites liées à la dynamique des modèles atomiques.

3.1.1 Problématiques de pondération

Pour garantir un partitionnement de qualité reflétant au mieux une subdivision de la simulation, il est nécessaire de parvenir à pondérer le graphe de modèles pour que celui-ci reflète la dynamique des modèles atomiques. Rappelons que d'un point de vue théorique, chaque sommet possède un poids reflétant le temps d'exécution du modèle atomique auquel il est rattaché et que chaque arc possède un poids qui reflète la quantité de messages en transit. Le temps d'exécution des modèles atomiques étant supposé identique ou négligeable, le poids des sommets est fixé à 1. Cependant, un modèle peut se composer de modèles atomiques comportant des dynamiques différentes. Il est donc naïf de penser que la quantité de messages en transit entre chaque modèle est identique. Cette quantité est totalement dépendante de la dynamique interne du modèle émetteur. Ces remarques

soulèvent une première interrogation : *Comment évaluer la fréquence de messages en transit entre deux modèles atomiques ?*

Une approche naïve serait d'exécuter la simulation et de comptabiliser le nombre de messages émis par chaque modèle atomique pour en déduire une fréquence d'émission. Cependant, cette démarche est un non sens, l'optimisation de la simulation distribuée ne peut pas dépendre d'une exécution de celle-ci. Cette remarque soulève une nouvelle question : *Comment déterminer cette fréquence sans réaliser la simulation complète ?*

La quantité de messages émis, ainsi que leur fréquence sont totalement dépendantes de la dynamique du modèle atomique émetteur. Or, seul le modélisateur a connaissance de la dynamique interne de chaque modèle atomique. Pour déterminer cette fréquence, il serait nécessaire d'observer la dynamique de chaque modèle atomique individuellement sans être forcé de réaliser toute la simulation, ainsi que les calculs qui lui sont attachés. *Comment observer cette dynamique à moindre frais ?*

L'idée défendue dans cette thèse repose sur la création d'un automate à états probabilistes donnant une représentation de la dynamique du modèle atomique qu'il reflète. L'intérêt d'une telle approche est de pouvoir générer des séquences d'observations statistiquement proches de celles du véritable modèle mais à moindre frais car ne nécessitant aucun calcul. Ces observations peuvent être vues comme des échantillons de la simulation et permettent donc de déterminer les fréquences nécessaires à l'élaboration de la pondération du graphe de modèles. La création de cet automate à états probabilistes se fait par un apprentissage individuel des dynamiques des modèles atomiques qui composent le modèle à simuler. Cependant, ces apprentissages soulèvent certaines questions comme par exemple : *Quelles sont les informations disponibles pour réaliser l'apprentissage ? Par quel moyen peut-on apprendre ces dynamiques ? Quel type d'automate à état probabiliste choisir ?*

3.1.2 Démarche d'apprentissage de la dynamique des modèles atomiques

Un modèle DEVS peut se composer d'un grand nombre de modèles atomiques dont les dynamiques peuvent varier. Dans notre problématique de pondération du graphe de modèles, il est nécessaire d'évaluer la fréquence d'émission de messages entre chaque couple de modèles atomiques. Or, cette fréquence est parfaitement dépendante de la dynamique du modèle atomique émetteur. Afin de déterminer chaque fréquence, il est nécessaire d'apprendre chacune des différentes dynamiques individuellement dans le but de créer un automate à états probabilistes pour chacune d'entre elle. L'apprentissage de cet automate se fait au travers de l'étude d'une base d'apprentissage fournie par le biais de la simulation d'un modèle atomique individuel.

Base d'apprentissage

Chaque modèle atomique possède une dynamique dont l'algorithme d'optimisation du partitionnement n'a pas connaissance. Ils peuvent être perçus comme une *"boîte grise"* car, même si la dynamique est inconnue, certaines informations sont tout de même accessibles. En effet, avant chaque transition interne la fonction λ est appelée pour effectuer une sortie. Même si la donnée

émise n'est pas connue, il est possible de déterminer le temps écoulé entre chaque appel de la fonction λ. Ces durées, notées d_i, reflètent la dynamique du modèle atomique car plus les d_i sont faibles plus le modèle effectue des sorties et inversement. En effet, ces durées sont un indicateur fiable permettant de déterminer la fréquence d'émission de messages entre les modèles atomiques.

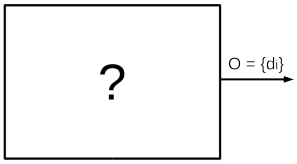


FIGURE 3.1 – Illustration de la notion de "boîte grise" comme représentant d'un modèle atomique

La simulation individuelle d'un modèle atomique durant un certain laps de temps fournit une séquence d'observations $O = \{d_1, d_2, \cdots, d_k\}$. Pour garantir la création d'un automate à états probabilistes proche de la dynamique du modèle atomique, il est nécessaire d'avoir des séquences d'observations suffisamment grandes, sans que le temps prit par la génération de celles-ci ne soit trop important. À partir de ces séquences d'observations et par application d'un algorithme d'apprentissage, un automate à états probabilistes est créé pour représenter la dynamique du modèle atomique. Mais qu'est ce qu'un automate à état probabiliste ?

Qu'est ce qu'un automate à état probabiliste ?

Un automate à état probabiliste est une construction abstraite, généralement sous forme d'un graphe orienté, se composant d'un nombre fini d'états. Chaque état décrit la configuration d'un système en attente d'exécuter une transition, où une transition est un ensemble d'actions à exécuter lorsqu'une condition est remplie ou lorsqu'un événement est reçu. La particularité des automates à état probabiliste est que chaque transition de l'automate est équipée d'une probabilité. Le passage d'un état à un autre s'effectue en fonction de ces probabilités. L'automate est une représentation de l'évolution du système au cours du temps. En effet, celui-ci est susceptible de se trouver dans chaque état mais seulement un à la fois. L'état dans lequel se trouve le système est appelé "état courant". La figure 3.2 est un exemple d'automate à état probabiliste à 3 états, où chaque état S_i peut effectuer une transition vers un état S_j avec une certaine probabilité ($P_{i,j}$).

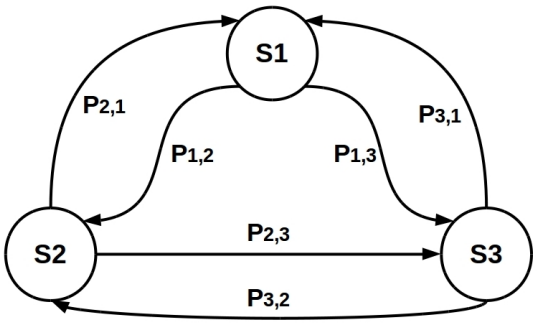


FIGURE 3.2 – Exemple d'automate à état probabiliste à trois états

La notion d'automate à état probabiliste a été introduite par Michael O. Rabin en 1963 dans [Rab63]. Il définit un automate à état probabiliste comme suit :

Définition 29 (Automate fini probabiliste) : Un automate fini probabiliste A est un quintuple $A = (S, \alpha, P, s_0, F)$ où

- S est l'ensemble fini des états
- α est l'alphabet fini d'entrée
- P est la matrice de transition des états associés à chaque entrée
- $s_0 \in S$ est l'état initial du système
- F est l'ensemble des états acceptants

Ce type de structure semble être parfaitement adéquat pour représenter la dynamique des modèles atomiques. Cependant, il est important de réaliser un apprentissage fiable pour garantir l'obtention de séquences statistiquement proches de celles obtenues par simulation. La qualité de la pondération du graphe de modèle repose entièrement sur la qualité de cet apprentissage, d'où l'importance de bien choisir la méthode.

Apprentissage et simulation

Le processus de pondération du graphe de modèles repose sur l'apprentissage de la dynamique que contiennent les modèles atomiques. Pour garantir un résultat optimal, il serait nécessaire d'apprendre la dynamique de chaque modèle atomique. Or, de part le temps des apprentissages lié à la taille du graphe de modèles et la création d'éventuels doublons, il n'est pas envisageable d'employer une telle démarche. Cette observation nous pousse à émettre l'hypothèse qu'un modèle comportant n modèles atomiques se compose de $N_d < n$ dynamiques différentes (contenues par les modèles atomiques). Ceci garantit l'existence de plusieurs modèles atomiques ayant la même dynamique. Pour chacune de ces dynamiques, on réalise une simulation individuelle d'un modèle atomique et on récolte la séquence d'observations des durées écoulées entre chaque émission de messages. Chaque séquence d'observations est considérée comme une base d'apprentissage à laquelle est appliquée une méthode d'apprentissage. Nous verrons, en section 3.1.3, que le choix de la méthode d'apprentissage est dépendant du type d'automate à états probabilistes que l'on souhaite obtenir et de son utilité ?

Chacun des N_p apprentissages donne naissance à un automate à état probabiliste qui reflète au mieux la dynamique du modèle appris. Ceci veut dire que lorsque l'on simule l'automate, la séquence d'observations obtenue doit être statistiquement proche de celle obtenue par simulation du modèle atomique. Attention, nous parlons bien d'une proximité "statistique", la séquence générée ne respectera certainement pas la logique du modèle mais seulement les statistiques d'apparition des différentes durées. Or, c'est cette seule information qui nous est nécessaire pour déterminer la fréquence d'émission de messages et par conséquent, le poids des arcs du graphe de modèles. À partir des ces automates à états probabilistes, on remplace chaque modèle atomique du graphe de modèles par l'automate qui représente sa dynamique. Ainsi, la simulation de ce graphe d'automates permet d'obtenir n séquences d'observations (une par modèle) à partir desquelles il est

possible de déterminer la fréquence d'émission moyenne de chaque modèle atomique. En considérant qu'une séquence d'observations contenant les durées entre deux émissions d'un modèle atomique se compose de k valeurs distinctes $d_1, d_2, \cdots, d_k$ qui se répètent dans la séquence, on note n_i $\forall i = 1, \cdots, k$ le nombre de fois où une durée d_i apparaît dans cette séquence. La durée moyenne d'émission entre deux sorties est donnée par :

$$d_{moy} = \frac{\sum_{i=1}^{k} d_i \times n_i}{\sum_{i=1}^{k} n_i}$$

On en déduit la fréquence moyenne d'émission par :

$$f_{moy} = \frac{1}{d_{moy}}$$

Pour chaque automate, et donc chaque dynamique, on obtient une fréquence moyenne f_{moy}. Ainsi, chaque arc sortant d'un sommet du graphe se voit attribuer la fréquence moyenne f_{moy} dépendamment de l'automate auquel est rattaché le sommet.

Pour les modèles atomiques sans entrée, l'intérêt de cette approche repose exclusivement sur le fait que l'apprentissage des N_p dynamiques est moins coûteux que la génération d'une séquence suffisamment grande pour garantir l'obtention d'une fréquence moyenne fiable par simulation. Cependant, nous verrons dans la section 3.3 que les modèles atomiques peuvent être soumis à des entrées. Pour respecter cette particularité, certaines modifications ont du être réalisées notamment en ce qui concerne la génération de séquences par simulation du graphe d'automates. Mais avant de nous intéresser plus en détail à cela, il est nécessaire de choisir une méthode d'apprentissage nous permettant d'atteindre nos objectifs.

3.1.3 Étude des méthodes d'apprentissage

La littérature offre différentes représentations des automates à état probabiliste. Leur nature, ainsi que leur création dépendent de l'approche d'apprentissage utilisée. Cette sous section a pour but de présenter les principaux automates à état probabiliste qu'offre la littérature, ainsi que les méthodes d'apprentissage permettant de les obtenir. Notre étude de ces automates se base sur le livre ***"Apprentissage artificiel, Concepts et algorithmes"*** de Antoine Cornuéjols et Laurent Miclet [CM10].

L'apprentissage peut être utilisé dans de nombreux domaines et pour répondre à certaines problématiques. En effet, l'apprentissage peut permettre d'optimiser un processus, de réaliser une prédiction ou encore de faire de la reconnaissance de forme. Ces apprentissages ont tous en commun de se baser sur l'environnement du processus. Il est également possible de classer les apprentissages suivants des problèmes plus abstraits comme la compression d'informations, la cryptographie ou encore l'analyse d'un phénomène. C'est à ce dernier que nous nous intéressons, il se rapporte à un problème d'approximation. Dans ce cas, la tâche de l'algorithme d'apprentissage est de trouver une approximation aussi bonne que possible d'un processus connu uniquement par

l'intermédiaire d'un échantillon de données.

La modélisation de la dynamique d'un système peut se faire par le biais de différents automates qu'il est possible d'apprendre. Notre attention s'est portée sur trois catégories d'automates, la première est le réseau de neurones. Un réseau neuronal est l'association, en un graphe plus ou moins complexe, de neurones formels. Les principaux réseaux se distinguent par leur architecture, leur niveau de complexité (nombre de neurones, $\cdots$), par le type de neurones et par l'objectif visé : apprentissage supervisé ou non, optimisation, etc $\cdots$ La caractéristique la plus intéressante d'un réseau de neurones est sa capacité à apprendre, c'est-a-dire de modifier les poids de ses connexions en fonction des données d'apprentissage. Ce type d'apprentissage se classe dans la catégorie de *"l'apprentissage de réseaux connexionniste"*. Même si l'automate obtenu est fiable en terme de représentation, il ne permet pas de simuler statistiquement un phénomène. Or, dans notre démarche, nous recherchons cette similitude statistique pour garantir l'obtention d'un automate proche de nos attentes.

Le second type d'automate que nous avons étudié est le réseau bayésien. Un réseau bayésien est un système de raisonnements probabilistes construit sur un graphe orienté sans cycle. Chaque nœud d'un réseau bayésien porte une étiquette qui est un des attributs du problème. Ces attributs sont binaires, pouvant prendre la valeur VRAI ou FAUX, ce qui signifie qu'une variable aléatoire est associée à chaque attribut et donc à chaque nœud. L'apprentissage consiste à trouver un réseau bayésien modélisant les données disponibles en s'appuyant éventuellement sur les connaissances à priori disponibles. Dans notre cas, le problème d'apprentissage se trouve dans la classe (données connues, structure inconnue) qui est l'une des quatre grandes familles de problèmes. Celui-ci se résout soit par le biais d'un algorithme de recherche discrète basé sur les contraintes qui consiste à tester les indépendances conditionnelles et à chercher une structure de réseau cohérente avec les dépendances et indépendances observées. Soit par le biais d'un algorithme de recherche discrète basé sur l'utilisation d'une fonction de score. Dans ce cas, un score est associé à chaque réseau candidat et l'on cherche le réseau maximisant ce score. Nous n'avons pas opté pour l'utilisation de cet automate car le jugeant trop restrictif. En effet, la nature sans cycle de son graphe empêche la répétition d'observations qui sont en réalité plausibles. Nous avons donc cherché un autre type d'automate permettant de générer des séquences d'observations quelconques, suivant certaines probabilités et offrant un algorithme d'apprentissage fiable pour son obtention. C'est notamment le cas des chaînes de Markov cachées présentées en section 3.2.

3.2 État de l'art sur les chaînes de Markov Cachées

Les chaînes de Markov cachées (HMM pour "Hidden Markov Model" en anglais) [Rab89] [BF96] sont des outils statistiques permettant de modéliser des phénomènes stochastiques. Elles sont utilisées dans de nombreux domaines tant pour la classification de données [SSVP02], la reconnaissance vocale [CHC12] et d'images [YOI92], que pour l'apprentissage de dynamiques [ESYD03]. Les chaînes de Markov cachées peuvent être considérées comme une amélioration des chaînes de Markov observables. Mais qu'est qu'une chaîne de Markov ?

3.2.1 Qu'est ce qu'une chaîne de Markov ?

Une chaîne de Markov, ou modèle de Markov observable, du nom de son inventeur Andreï Markov, est un processus stochastique reposant sur la propriété que *"la prédiction du futur s'effectue par le biais d'informations exclusivement contenues dans l'état présent du processus et ne dépend pas des états antérieurs"*. On dit d'un tel système qu'il n'a pas de "mémoire", seul l'état courant influe sur l'état futur. Une chaîne de Markov est un processus aléatoire dont les états S_i peuvent changer au cours du temps. L'observation de l'évolution de ces états forme des séquences d'états $S = S_1, \cdots, S_n$. Chaque séquence est émise avec une certaine probabilité $P(S) = P(S_1, \cdots, S_n)$ qu'il est possible de calculer à condition de connaître la probabilité de l'état initial $P(S_1)$ et les probabilités d'être dans un état S_t tout en connaissant les évolutions antérieures. La nature stochastique des chaînes de Markov repose sur l'évolution du système en fonction d'une probabilité initiale et des probabilités de transition entre états. Ce qui s'exprime mathématiquement par :

$$P(q_t = S_i | q_{t-1} = S_j, S_{t-2} = S_k, \cdots) = P(q_t = S_i | q_{t-1} = S_j)$$

où q_t est l'état observé à l'instant t et S_i un état du système.

Dans notre cas, les chaînes de Markov sont dites stationnaires, c'est à dire que les transitions entre états n'évoluent pas au cours du temps. Cette condition permet de définir une matrice de transition $A = [a_{i,j}]$ tel que

$$a_{i,j} = P(q_t = s_j | q_{t-1} = s_i) \text{ tel que } \ 1 \leq i, j \leq n$$

où $a_{i,j}$ est la probabilité que l'état courant soit l'état j alors qu'on était dans l'état i au pas de temps précédent. Cette matrice a la particularité de respecter les conditions suivantes :

$$\forall i, j \ \ a_{i,j} \geq 0 \ \text{ et } \ \forall i \sum_{j=1}^{j=n} a_{i,j} = 1$$

Les chaînes de Markov ne sont pas nouvelles, les premiers résultats concernant un espace d'états fini d'Andreï Markov date de 1906. Par la suite, une généralisation à un espace d'états infini dénombrable a été publiée par Kolmogorov en 1936.

3.2.2 Qu'est ce qu'une chaîne de Markov cachée ?

Une chaîne de Markov cachée se compose de deux processus stochastiques : l'un caché et l'autre observable.

- le processus caché est modélisé par une chaîne de Markov observable. Dans ce cas, la suite des états observés $q_1, q_2, \cdots, q_T$ est cachée à l'utilisateur.

- le processus observé dépend des états du processus caché. Chaque état caché émet une observation O_i appartenant à un alphabet de M symboles observables $V = \{v_1, v_2, \cdots, v_M\}$

Contrairement aux chaînes de Markov observables, chaque état d'un HMM n'est pas associé à un unique symbole mais à un alphabet de symboles noté V. Cet alphabet est commun à chaque état,

il est donc possible pour chaque état d'émettre un symbole de cet alphabet avec une certaine probabilité. Ce n'est donc pas les états qui sont observés mais les symboles qu'ils émettent. L'intérêt de cette approche est qu'il n'y a pas de limite de taille pour l'alphabet. Dans cette représentation, chaque état possède sa propre densité de probabilité pour la distribution des symboles.

Définition 30 (Chaîne de Markov Cachée) : Une chaîne de Markov cachée se formalise par le biais du triplé $\lambda = (\pi, A, B)$ où

- $\pi = \{\pi_1, \cdots, \pi_N\}$ est le vecteur des probabilités initiales. Pour tout état i, π_i est la probabilité que l'état de départ du HMM soit l'état i, $\pi_i = P(q_t = S_i)$.
- $A = \{a_{i,j} |\ i, j = 1, \cdots, N\}$ est la matrice de probabilité de transition entre les états.
- $B = \{b_{i,j} |\ i = 1, \cdots, N\ ;\ j = 1, \cdots, M\}$ est la matrice de probabilité d'observation des symboles dans chacun des états. Où, $b_{i,j}$ est la probabilité d'observer le symbole v_j alors que le système se trouve dans l'état S_i.
- N est le nombre d'états du système. Les états sont définis par $S = \{S_1, \cdots, S_N\}$.
- M est le nombre de symboles contenus dans l'alphabet $V = \{v_1, \cdots, v_M\}$

Les matrices de probabilité π, A et B sont construites pour respecter les conditions suivantes :

$$\sum_{j=1}^{N} \pi_j = 1\ ,\ \sum_{j=1}^{N} a_{i,j} = 1 \text{ et } \sum_{j=1}^{M} b_{i,j} = 1\ \ \forall i \in [1, N]$$

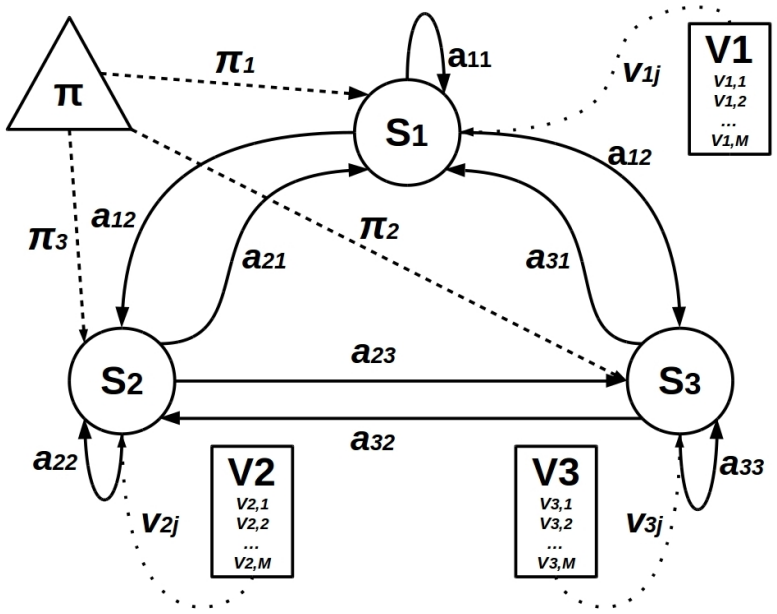


FIGURE 3.3 – Graphe de dépendances d'un HMM à 3 états et M symboles

Il est possible de représenter un HMM à partir d'un graphe de dépendances entre la séquence des états cachés S et la séquence des observations O. La figure 3.3 donne un exemple de graphe de dépendances pour une chaîne à 3 états. Dans notre utilisation des HMM, nous supposons qu'il n'existe aucune contrainte concernant la transition des états. En effet, toutes les transitions d'un état vers un autre sont possibles. Par définition, un tel HMM est dit ergodique. La figure 3.3 est un exemple de chaîne de Markov cachée ergodique.

L'utilisation des chaînes de Markov cachées offre une flexibilité que n'ont pas les chaînes de Markov observables. En effet, pour les chaînes observables chaque état est contraint à un unique symbole, ce qui signifie forcement que sa représentation offre autant d'états qu'il y a de symboles observables. Cet aspect contraint la modélisation du système. Or, pour les chaînes cachées, chaque état a accès à un alphabet de symboles qu'il émet avec une certaine probabilité $b_{i,j}$. Il n'est donc plus nécessaire d'avoir autant d'états qu'il y a de symboles à observer. La flexibilité du nombre d'états cachés permet la création d'un automate dont la génération de séquences tend vers la réalité.

3.2.3 Les trois problématiques liées aux HMM

L'utilisation des HMM s'oriente autour de trois grandes problématiques :

— Le problème de l'inférence : étant donné une chaîne et une séquence d'observations, quelle est la séquence d'états cachés la plus probable d'avoir généré cette séquence d'observations ?

— Le problème de reconnaissance ou de classement : étant donné une liste de chaînes et une séquence d'observations, on cherche à sélectionner la chaîne la plus optimale, c'est à dire celle ayant la plus forte probabilité d'avoir généré cette séquence.

— Le problème d'apprentissage : étant donnée une séquence d'observations O, comment apprendre les paramètres $\lambda = (\pi, A, B)$ d'un HMM pour maximiser $p(O|\lambda)$?

Le problème de l'inférence se rapporte à déterminer la séquence d'états $S = S_1, S_2, \cdots, S_k$ la plus probable ayant pu générer la séquence d'observation $O = o_1, o_2, \cdots, o_k$. Une méthode globale pour résoudre ce problème serait de déterminer toutes les séquences d'états ayant pu générer O, puis calculer leur probabilité afin de déterminer la plus probable. Le problème de cette approche est qu'elle est très gourmande en calculs car totalement dépendante du nombre d'états du système. La littérature offre une méthode pour résoudre ce genre de problème : l'algorithme de Viterbi présenté dans [Vit67] par l'auteur du même nom. Les explications pour la résolution de ce problème se limitent à cette seule référence car il ne fait pas l'objet de cette thèse. De même pour le problème de reconnaissance, la résolution de celui-ci peut se faire notamment par le biais de l'algorithme du Forward présenté dans [HAJ90].

Le problème d'apprentissage correspond parfaitement à notre problématique d'apprentissage de la dynamique. En effet, nous cherchons à créer un automate à état probabiliste à partir d'une séquence d'observations O correspondant aux différentes durées d_i entre chaque émission de sortie. En supposant que ces d_i forment l'alphabet de symboles V de notre HMM, il est possible de créer un HMM à N états en apprenant les paramètres du système $\lambda = (\pi, A, B)$ de telle sorte que

le HMM résultant fournisse des séquences d'observations statistiquement proches de celle de la base d'apprentissage O.

L'apprentissage

Partant d'un ensemble de séquences $\theta = \{O^1, O^2, \cdots, O^k\}$, il est possible d'apprendre les paramètres d'un HMM $\lambda = (\pi, A, B)$ de façon à maximiser $P(\theta|\lambda)$. Comme les séquences formant la base d'apprentissage sont supposées indépendantes, on cherche à maximiser

$$P(\theta|\lambda) = \prod_{i=1}^{k} P(O^i|\lambda)$$

En partant d'une initialisation des paramètres d'un HMM $\lambda_0 = (\pi_0, A_0, B_0)$, la procédure consiste à ré-estimer ces paramètres de façon à satisfaire $P(\theta|\lambda_{i+1}) \geq P(\theta|\lambda_i)$. Le respect de cette contrainte garantit une amélioration de la probabilité d'émission des observations des symboles de la base d'apprentissage θ. La ré-estimation des paramètres peut se faire à l'aide de l'algorithme de Baum-Welsh présenté dans [Bil98], qui est dérivé de l'algorithme EM (Expectation Maximization) présenté dans [BC94].

Cet algorithme entreprend une ré-estimation des paramètres d'un HMM de manière à augmenter la vraisemblance de génération des séquences d'observations. La maximisation de la vraisemblance $P(\theta|\lambda)$ peut donc se voir comme l'optimisation du HMM λ. Celle-ci doit se faire rapidement, pour cela il est nécessaire d'utiliser un algorithme de calcul de complexité faible. Le fonctionnement de l'algorithme de Baum-Welsh nécessite l'application des algorithmes ***Forward*** et ***Backward*** pour déterminer respectivement les informations suivantes $P(\theta|\lambda)$, $\alpha_t(i)$ et $\beta_t(i)$. $\alpha_t(i)$ est la probabilité de la suite d'observations partielles $(o_1\ o_2\ \cdots\ o_t)$, se terminant à l'état S_i à l'instant t

$$\alpha_t(i) = P(o_1\ o_2\ \cdots\ o_t,\ q_t = S_i|\lambda)$$

$P(\theta|\lambda)$ est la probabilité d'apparition des séquences observées $\theta = (O^1, O^2, \cdots, O^k)$, $P(\theta|\lambda)$ est telle que :

$$P(\theta|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

et $\beta_t(i)$ peut se voir comme la probabilité d'observer la suite partielle $(o_{t+1}\ o_{t+2}\ \cdots\ o_T)$, qui commence à l'instant $t+1$ et dont l'état à l'instant t est S_i

$$\beta_t(i) = P(o_{t+1}\ o_{t+2}\ \cdots\ o_T,\ q_t = S_i|\lambda)$$

$P(\theta|\lambda)$ peut également s'obtenir à partir de $\beta_t(i)$ puisque

$$P(\theta|\lambda) = \sum_{i=1}^{N} \beta_1(i)$$

Le rôle de l'algorithme ***Forward***, présenté par l'algorithme 20, est de calculer itérativement la matrice α car l'information qu'elle condense est nécessaire au déroulement de l'algorithme de

Baum-Welsh.

Algorithme 20 Algorithme Forward

PROCÉDURE: Forward
ENTRÉES: $\lambda_i = (\pi_i, A_i, B_i)$, $O = o_1\ o_2\ \cdots\ o_T$
  Pour i allant de 1 à N **faire**
    $\alpha_1(i) = \pi_i b_i(o_1)$
  Fin pour
  Pour t allant de 1 à $T-1$ **faire**
    Pour j allant de 1 à N **faire**
      $\alpha_{t+1}(j) = (\sum_{i=1}^{N} \alpha_t(i) a_{ij})\ b_j(o_{t+1})$
    Fin pour
  Fin pour
  $P(\theta|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$
  Sorties: $P(\theta|\lambda)$, α

L'algorithme ***Backward***, présenté par l'algorithme 21 est conçu de manière similaire à l'algorithme ***Forward*** dans son fonctionnement. Son rôle est de calculer itérativement la matrice β qui est également indispensable au déroulement de l'algorithme de Baum-Welsh.

Algorithme 21 Algorithme Backward

PROCÉDURE: Backward
ENTRÉES: $\lambda_i = (\pi_i, A_i, B_i)$, $O = o_1\ o_2\ \cdots\ o_T$
  Pour i allant de 1 à N **faire**
    $\beta_T(i) = 1$
  Fin pour
  Pour t allant de $T-1$ à 1 **faire**
    Pour i allant de 1 à N **faire**
      $\beta_t(i) = \sum_{j=1}^{N} a_{ij}\ \beta_{t+1}(j)\ b_j(o_{t+1})$
    Fin pour
  Fin pour
  Sorties: β

A partir des variables $\alpha_t(i)$ et $\beta_t(i)$, il est possible de calculer la vraisemblance $P(\theta|\lambda)$ des séquences d'observations θ, pour le HMM λ, à chaque instant t :

$$P(\theta|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\ \beta_t(i)$$

Les estimations du HMM $\lambda = (\pi, A, B)$ effectuées par l'algorithme de Baum-Welsh sont conçues ainsi :

$$\overline{\pi}_i = \text{probabilité d'être dans l'état } S_i \text{ à l'instant } t = 1$$
$$\overline{a}_{ij} = \frac{\text{nombre de transitions de l'état } S_i \text{ vers } S_j}{\text{nombre de fois où on quitte } S_i}$$

$$\overline{b}_j(k) = \frac{\text{nombre de fois où l'on est dans l'état } S_i \text{ en observant le symbole } v_k}{\text{nombre de fois où l'on est dans l'état } S_j}$$

Pour estimer ces nouvelles probabilités, l'algorithme de Baum-Welch utilise deux nouvelles matrices : ε et γ. Les coefficients de ε, $\varepsilon_t(i,j)$ représentent la probabilité d'être dans l'état S_i à l'instant t et de passer dans l'état S_j à l'instant $t+1$. Et les coefficients de γ, $\gamma_t(i)$ représentent la probabilité d'être dans l'état S_i à l'instant t. Le calcul de ces coefficients dépend des matrices α et β obtenues grâce aux algorithmes Forward et Backward. Le calcul des coefficients $\varepsilon_t(i,j)$ s'obtient par

$$\varepsilon_t(i,j) = \frac{\alpha_t(i)\, a_{i,j}\, b_j(o_{t+1})\, \beta_{t+1}(j)}{P(\theta|\lambda)}$$

et ceux de γ comme suit

$$\gamma_t(i) = \sum_{j=1}^{N} \varepsilon_t(i,j) = \frac{\alpha_t(i)\, \beta_t(i)}{P(\theta|\lambda)}$$

La ré-estimation des paramètres π, A et B du HMM λ s'obtient par :

$$\overline{\pi}_i = \gamma_1(i) \quad 1 \leq i \leq N$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \varepsilon_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad 1 \leq i,j \leq N$$

$$\overline{b}_j(k) = \frac{\sum_{t=1 \cap o_t = v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Algorithme 22 Algorithme de Baum-Welsh

PROCÉDURE: Baum-Welsh
ENTRÉES: $\lambda = (\pi, A, B)$, θ
  Tantque $P(\theta|\lambda)$ peut être augmentée **faire**
    $(P(\theta|\lambda), \alpha) =$ **Forward**(λ, θ)
    $\beta =$ **Backward**(λ, θ)
    Pour t allant de 1 à T **faire**
      Pour i allant de 1 à N **faire**
        Pour j allant de 1 à N **faire**
          Calcul de $\varepsilon_t(i,j)$
        Fin pour
        Calcul de $\gamma_t(i)$
      Fin pour
    Fin pour
    Ré-estimation de $\lambda = (\pi, A, B)$
  Fin Tantque
  Sorties: λ

Suite à chaque ré-estimation de paramètres, l'algorithme recalcule la vraisemblance. Le processus est réitéré tant que la vraisemblance n'est pas maximale. L'algorithme 22 présente le fonctionnement de l'algorithme de Baum-Welsh. La qualité de l'apprentissage d'un HMM par l'algo-

rithme de Baum-Welsh est dépendante de l'initialisation des paramètres $\lambda_0 = (\pi_0, A_0, B_0)$. En effet, le choix des paramètres initiaux est primordial, si l'on fixe une probabilité à 0 celle-ci reste égale à 0 tout au long de l'apprentissage. Il est conseillé de donner des probabilités quasi-équiprobables à l'initialisation. Mais si toute fois, certaines informations sont disponibles *a priori*, il est bon de les prendre en compte afin d'accélérer le processus d'apprentissage.

3.3 Adaptation de l'apprentissage des HMM aux contraintes des modèles DEVS

Le choix de la modélisation de la dynamique d'un modèle DEVS par un HMM n'est pas anodin. En effet, si l'on compare les structures de chacune de ces entités, on peut constater qu'elles sont similaires en bien des points. Premièrement, elles sont toutes deux munies d'un système à états qui évolue au cours du temps. Certes, le changement d'état ne se produit pas de la même manière : pour le modèle DEVS, ce changement est déterministe (en général, même si un modèle DEVS peut être stochastique) et exclusivement lié à la fonction d'avancement du temps t_a, alors que pour un HMM, le changement d'état est purement probabiliste. Cependant en vue de notre problématique de génération de séquences statistiquement proches, cet aspect n'a pas d'importance. De plus, elles ont en commun d'émettre une sortie à chaque changement d'état. Pour les modèles DEVS, cette remarque est exclusivement valable lors d'un changement d'état par transition interne. En effet, lors d'une transition externe, le modèle change d'état sans effectuer de sortie. Afin de modéliser au mieux la dynamique d'un modèle DEVS à l'aide d'un HMM, il est nécessaire de prendre en compte les particularités liées au formalisme DEVS lors de la création du HMM. Ceci implique un certain nombre de modifications, tant sur la modélisation du HMM lui-même que sur les algorithmes de génération de séquences.

3.3.1 Politiques de génération des bases d'apprentissage en fonction des entrées d'un modèle DEVS

Afin d'obtenir une pondération de qualité, il est important de bien apprendre chacune des dynamiques qui composent le modèle. Supposons qu'un modèle DEVS se compose de modèles atomiques dont la dynamique qui les anime est identique. *Sommes nous en droit de nous demander si un unique apprentissage suffirait à comprendre la dynamique totale du modèle ?* La réponse à cette question se trouve dans ce simple contre exemple : supposer que la dynamique des modèles est parfaitement identique revient à apprendre un unique HMM et à remplacer le graphe de modèle par un graphe de HMM, où chaque HMM est identique. De par la nature probabiliste des HMM, les séquences d'observations obtenues par simulation du graphe de HMM devrait comporter les mêmes fréquences moyennes d'émission de messages et donc fournir par conséquent une pondération identique sur chaque arc. Ce qui équivaudrait à une pondération unitaire et serait parfaitement inutile.

Ce contre exemple montre que pour une même dynamique "de base", les modèles atomiques peuvent avoir un comportement différent. *Quelle peut être l'origine de cette "déformation" de la*

dynamique, pourtant initialement la même ?

Influence du nombre d'entrées d'un modèle atomique sur sa dynamique

Rappelons que, dans notre contexte DEVS, il existe deux types de modèles atomiques : les modèles sans entrée et les modèles avec entrées. L'évolution des états du système du modèle sans entrée est exclusivement liée à la dynamique de celui-ci, qui est régie par la fonction de transition interne δ_{int} et la fonction d'évolution du temps t_a. Dans le cas des modèles avec entrées, la dynamique du système se voit perturbée par l'apparition d'événements externes. En effet, si un événement apparaît au temps $e = t_2 < t + ta(s)$, la fonction de transition externe est appelée pour déterminer le nouvel état. Or la transition interne, et donc la sortie, qui aurait dû avoir lieu au temps $t + t_a(s)$ n'a pas lieu ce qui modifie la dynamique du système. Cette remarque nous porte à croire que deux modèles atomiques identiques mais n'ayant pas le même nombre d'entrées ne sont pas soumis à la même trajectoire. Ceci nous pousse à émettre l'hypothèse que *"le nombre d'entrées d'un modèle atomique influence sa dynamique"*, en général. Il peut toujours exister des modèles dont la fonction de transition externe n'impacte pas les trajectoires.

En partant de cette hypothèse, et en supposant que chaque modèle atomique est soumis à la même dynamique "de base", il est nécessaire de réaliser l'apprentissage d'un HMM par nombre d'entrées. La figure 3.4 illustre cette hypothèse d'apprentissage. Dans ce cas, chaque HMM est indexé par rapport au nombre d'entrées que contient le modèle atomique auquel il est attaché.

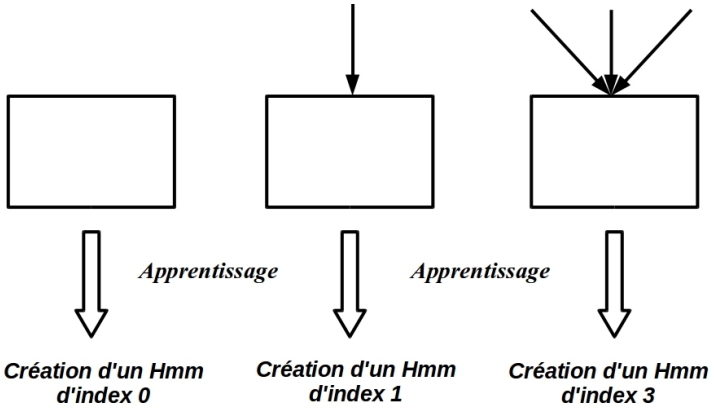


FIGURE 3.4 – Influence du nombre d'entrées des modèles atomiques qui compose le modèle DEVS sur le nombre d'apprentissages

L'apprentissage de ces N_{app} HMM nécessite la création de N_{app} bases d'apprentissage (une pour chaque type d'entrées). La génération de la séquence sans entrée se fait par une simple simulation d'un modèle atomique sans entrée durant un laps de temps t_{max}. Cependant, en ce qui concerne les modèles avec entrées, comment générer une séquence d'observation ? La question est légitime, qui dit modèle avec entrées dit séquence d'entrée. Comment générer ces entrées ?

Les générateurs d'entrées

Rappelons que, dans le cadre de l'apprentissage, nous ne nous intéressons pas à la valeur de l'entrée mais seulement à la date à laquelle a lieu cette entrée. La notion de séquence d'entrée correspond donc aux différentes dates auxquelles le modèle atomique est soumis à une entrée.

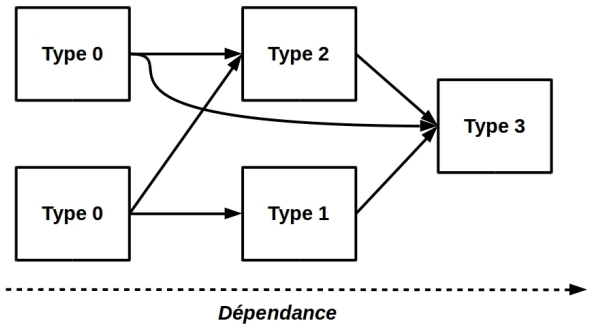


FIGURE 3.5 – Illustration du lien de dépendance entre les modèles avec entrées et les modèles sans entrée

La première hypothèse, et probablement la plus "pauvre", consiste à générer un spectre de dates d'entrée le plus large possible. Cette hypothèse nécessite la création d'un générateur de durées entre 2 entrées comprises entre deux valeurs d_{min} et d_{max}, où d_{min} correspond à la plus petite durée entre deux arrivées d'un message et d_{max} à la plus grande. L'intérêt de cette approche est d'explorer toutes les configurations possibles. Cependant, elle a aussi le désavantage de prendre en compte des cas irréalistes, car n'existant pas dans la simulation du modèle DEVS. Pour limiter cet effet, il est possible de paramétrer d_{min} et d_{max} à partir des valeurs d'une séquence d'observations générées par un modèle sans entrée. Dans ce cas, d_{min} correspond à la plus petite durée entre 2 arrivées de la séquence sans entrée et d_{max} à la plus grande. En effet, il est simple d'envisager que toutes les entrées des modèles sont forcement dépendantes, à un moment ou à un autre, d'une séquence générée par un modèle sans entrée. La figure 3.5 illustre cette hypothèse par un exemple. On peut voir que toutes les séquences générées par un modèle avec entrée ont forcément eu besoin d'une séquence générée par un modèle sans entrée à un moment de la simulation.

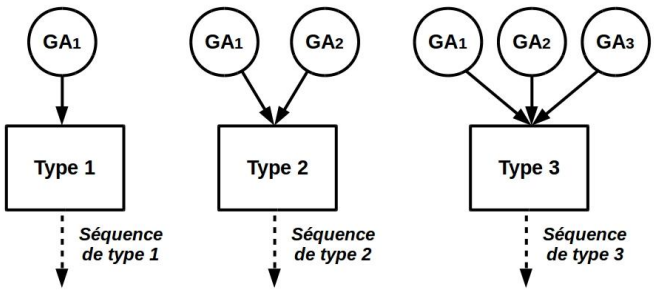


FIGURE 3.6 – Illustration de l'utilisation des générateurs purement aléatoires GPA

Ce générateur d'entrées, appelé **générateur purement aléatoire (GPA)**, peut être conçu comme un modèle atomique DEVS sans entrée où la fonction t_a est un tirage aléatoire d'une valeur comprise entre d_{min} et d_{max}. Ainsi lorsque $e = t_a(s)$ la fonction de sortie est appelée pour émettre la valeur du $t_a(s)$ courant : $\lambda(s) = t_a(s)$. La génération des séquences d'apprentissage par application

d'un générateur aléatoire est illustrée par la figure 3.6. À chaque entrée d'un modèle atomique est associé un modèle de type **GPA** dont la dynamique simule des sorties aléatoires.

Poussons d'avantage l'utilisation de la séquence d'observations sans entrée seq_0 pour la création d'un générateur d'entrées plus intuitif. En effet, en reprenant l'hypothèse émise sur les liens de dépendance entre les séquences d'observations sans entrée et la dynamique des modèles avec entrées, il est possible d'optimiser l'utilisation des séquences seq_0 pour garantir la création de séquences d'apprentissage plus proches de la réalité pour les modèles à entrées multiples. Chaque séquence seq_0 est analysée dans le but de connaître les différentes durées d_i qu'elle contient. Si l'on suppose que ces d_i sont les seules entrées que peut recevoir un modèle, il serait judicieux d'améliorer le générateur d'entrées de façon à ce qu'il ne génère aléatoirement que des d_i appartenant à seq_0. La création d'un tel générateur, que l'on appellera **générateur séquence aléatoire (GSA)**, repose dans un premier temps sur l'analyse d'une séquence seq_0 afin de déterminer l'ensemble des $d_i \in seq_0$. Dans un second temps, le **GSA** réalise un tirage aléatoire uniforme parmi les $d_i \in seq_0$ pour simuler les entrées que peuvent recevoir les modèles. Il est possible d'améliorer le procédé en déterminant la probabilité d'occurrence $P(d_i)$ de chaque d_i de la séquence seq_0 et de réaliser un tirage respectant ces probabilités pour générer une séquence d'entrées statistiquement proche de seq_0. L'implémentation de ce générateur se fait par le biais d'un modèle atomique, où la fonction t_a est un tirage d'un $d_i \in seq_0$ respectant les probabilités $P(d_i)$. De même que pour le modèle atomique **GPA**, la génération se fait à l'aide de la fonction de sortie λ appelée avant chaque transition interne lorsque $e = t_a(s)$ (s est l'état courant du système). La figure 3.7 schématise le processus de création du générateur **GSA**, ainsi que son fonctionnement.

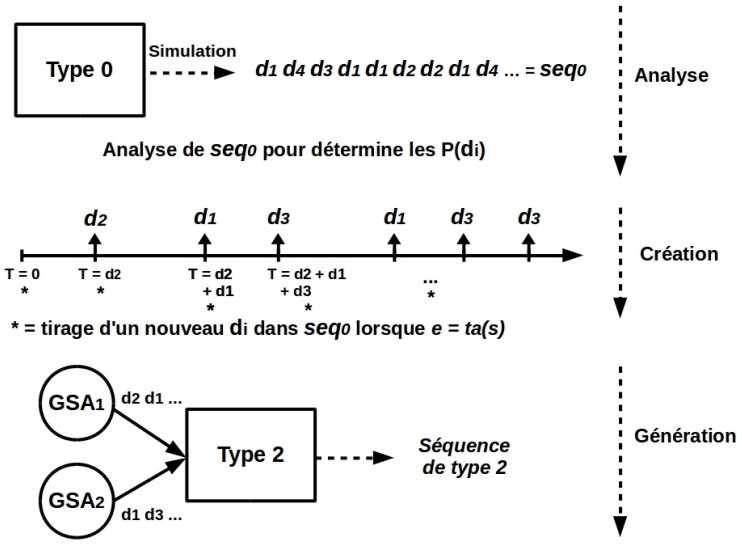


FIGURE 3.7 – Illustration de la création et de l'utilisation de générateurs GSA

Le dernier générateur que nous avons créé génère des entrées identiques à la séquence seq_0. En repartant de l'idée que tout modèle à entrées multiples dépend un minimum d'une séquence d'observations sans entrée, nous avons décidé de tester un apprentissage où les entrées des modèles atomiques sont fournies par la séquence seq_0. Ce générateur porte le nom de **générateur par séquence sans entrée (GSSE)**.

La validation de ces générateurs, notamment en ce qui concerne leur impact sur la qualité de l'apprentissage, est présentée en section 3.4. Mais avant d'étudier cette validation, il est important de bien récapituler les différentes phases de l'apprentissage des dynamiques à l'aide des chaînes de Markov cachées.

3.3.2 Récapitulatif de l'apprentissage des dynamiques à l'aide des HMM

L'apprentissage de la dynamique d'un modèle atomique nécessite la création d'une base d'apprentissage. Celle-ci correspond aux différentes durées d_i d'émission entre deux envois de messages. Cette séquence d'apprentissage est obtenue par simulation d'un modèle atomique indépendant. Pour les modèles atomiques sans entrée, une simple simulation durant un certain laps de temps suffit à générer cette séquence d'apprentissage, nommée seq_0 dans ce cas. Les modèles atomiques avec entrées nécessitent l'utilisation de générateur d'entrées, présentés dans la sous-section précédente, pour simuler les entrées nécessaires à l'exécution du modèle. Ces simulations indépendantes donnent naissance à des séquences d'apprentissage de la forme

$$d_1\ d_4\ d_7\ d_7\ d_2\ d_1\ d_1\ d_2 \infty\ d_3\ d_4\ d_1\ d_6\ d_5\ d_3\ d_2\ d_6\ \cdots$$

Rappelons qu'un modèle atomique peut se trouver dans un état "infini", c'est à dire que $t_a = \infty$ et que le modèle reste en sommeil tant qu'aucun événement extérieur ne vient le perturber. Il se peut donc qu'un d_i prenne la valeur ∞ dans ce cas. Or, la valeur ∞ n'est pas à considérer en tant que durée, mais plutôt comme un indicateur sur l'état courant du modèle atomique. Ce symbole, exclusivement lié à la dynamique DEVS, est indispensable au bon déroulement des algorithmes de génération de séquences présentés en section 3.3.3. La simulation individuelle des modèles atomiques donne des séquences d'observations déterministes, c'est à dire que deux simulations indépendantes fournissent systématiquement le même résultat. Or, l'apprentissage nécessite plusieurs séquences d'observations pour garantir sa convergence. Il serait parfaitement inutile de simuler n fois un même modèle, cela fournirait n fois la même séquence. Nous proposons de réaliser une "longue" simulation d'un modèle atomique dans le but d'obtenir une séquence d'apprentissage de grande taille. Ainsi cette séquence peut être subdivisée et utilisée comme base d'apprentissage. Nous avons opté pour une subdivision en 20 sous-séquences de tailles différentes.

Pour chaque type de modèle (en fonction du nombre d'entrées), on initialise un HMM $\lambda = (\pi, A, B)$ dont le nombre d'états est fixé à $N+1$, comprenant un unique état final. La littérature offre des moyens permettant de déterminer le nombre optimal d'états N garantissant un apprentissage optimal, comme par exemple [BSVdB97]. Cependant, ces méthodes sont souvent gourmandes en temps de calculs et risqueraient de pénaliser le temps de simulation. Nous proposons une approche simple et à moindre de frais pour déterminer ce nombre d'état. Celle-ci consiste à apprendre N HMM à partir d'une séquence obtenue sur un modèle sans entrée, où le nombre d'états qui les composent varient entre 2 et N. La séquence sans entrée étant la plus fiable pour réaliser une comparaison, cela nous permet de déterminer le nombre d'états optimal pour garantir un HMM offrant une erreur minimale. L'alphabet V des symboles correspond aux différentes durées d_i observées dans la séquence d'apprentissage, incluant le symbole ∞. À chaque symbole est

associé une probabilité d'émission par état $b_{i,j}$, supposée initialement équiprobable. Dans le cas de l'apprentissage des modèles sans entrée, la probabilité $b_{i,\infty}$ est nulle et le restera tout au long de l'apprentissage. Chaque état $S_i \; \forall \; i < 6$ différent de l'état final peut être choisi comme état initial du système, ce qui implique une probabilité $\pi_i \; i < 6$ équiprobable et $\pi_6 = 0$. De plus, le HMM étant supposé ergodique, il est possible de passer d'un état S_i à n'importe quel autre état S_j avec une certaine probabilité $a_{i,j}$. Or à l'initialisation ces transitions sont supposées équiprobables. Pour favoriser une convergence rapide de l'algorithme d'apprentissage, il est préférable d'initialiser les différentes probabilités de manière "quasi-équiprobable". Cette perturbation peut être vue comme un moyen de s'extraire d'un éventuel maximum local.

À partir des sous-séquences d'apprentissage et du HMM initial $\lambda_0 = (\pi_0, A_0, B_0)$, les paramètres π, A et B sont ré-estimés par application de l'algorithme de Baum-Welsh. Cet algorithme détermine les meilleurs π, A et B susceptibles de favoriser la génération d'une séquence d'observations statistiquement proche de celles fournies par la base d'apprentissage. D'où l'importance d'avoir des séquences d'apprentissage très proches de la réalité du modèle atomique. Une fois chaque HMM appris, il est possible de remplacer le graphe de modèles par un graphe de HMM afin d'en faire une simulation. La simulation de celui-ci permet de générer des séquences d'observations sans être soumis aux calculs liés aux modèles atomiques. Pour chaque modèle atomique, on détermine la durée moyenne d'émission d_{moy} à partir de ces séquences d'observations. Lors de son calcul, le symbole ∞ est ignoré car il n'est pas une durée mais plutôt un indicateur de l'état courant du modèle ("en attente" si $d_i = \infty$, "actif" sinon). La pondération du graphe de modèles s'effectue à partir de la fréquence moyenne d'émission de message correspondant à $\frac{1}{d_{moy}}$. En effet chaque arc sortant d'un modèle atomique est pondéré par la fréquence d'émission moyenne de celui-ci.

Une simulation peut se composer de modèles atomiques possédant K dynamiques différentes. Chacun de ces modèles atomiques peut comporter entre 0 et n entrées, ce qui implique au maximum $K \times n$ apprentissages pour déterminer toutes les dynamiques d'un modèle DEVS. Une fois ces dynamiques apprises, chaque modèle atomique du graphe de modèle est remplacé par son représentant HMM, dans le but de réaliser une simulation sans être pénalisé par le temps de calcul des modèles atomiques. Dans le cadre de notre expérimentation, les K dynamiques de base sont supposées connues et sont donc une information inclue dans chacune des "boîtes grises". Plus précisément, nous nous autorisons à connaître une pseudo classification des modèles suivant leur dynamique, par le biais d'un paramètre de typage. La pertinence de cette approche repose sur l'hypothèse que le temps de calcul des modèles atomiques est nettement plus important que le temps d'apprentissage des dynamiques.

3.3.3 Génération de séquences d'observations à partir du graphe de HMM

La simulation du graphe de HMM repose sur l'application d'un algorithme de génération de séquence d'observations sur chaque HMM qui compose le graphe. L'algorithme de génération de séquences standards des HMM est initialement conçu pour générer des séquences d'observations en se basant exclusivement sur l'alphabet de symbole V ainsi que sur la matrice de probabilité d'émission de symbole B. En effet, cet algorithme est en réalité une procédure itérative gérée par

des tirages aléatoires. Il peut être utilisé pour la génération de séquences de HMM de type 0, c'est à dire dont le HMM provient d'un modèle sans entrée. Il est vrai que dans ce cas, l'algorithme de génération de séquences standards des HMM suffit.

Algorithme 23 Algorithme de génération de séquences d'observations pour un HMM représentant un modèle sans entrée

PROCÉDURE: Générateur standard
ENTRÉES: $\lambda = (\pi, A, B)$
Tirage d'un état initial S_i suivant les probabilités π
$S_{courant} \leftarrow S_i$ devient l'état courant
Tirage d'un symbole $v_k \in V$ suivant les probabilités $b_{i,k}$
$seq \leftarrow v_k$ ajout du symbole tiré à la séquence d'observations
Tantque $S_{courant} \neq S_{final}$ **faire**
Tirage d'un état futur S_j suivant les probabilités a_{ij}
$S_{courant} \leftarrow S_j$ devient l'état courant
Tirage d'un symbole v_k suivant les probabilités b_{ij}
$seq \leftarrow seq \cup \{v_k\}$ ajout du symbole tiré à la séquence d'observations
Fin Tantque
Sorties: seq

La création du graphe de HMM génère l'apparition de connexions entre les HMM qui le compose. Pour l'algorithme 23, les séquences peuvent être générées sans problème car il ne dépend pas d'informations extérieures. Lors de la simulation d'un modèle DEVS, les modèles à entrées multiples reçoivent des événements de la part de modèles émetteurs. Certains modèles à entrées multiples restent dans un état "infini" tant qu'aucun événement ne vient les perturber. Ces particularités liées au formalisme DEVS doivent être prises en compte lors de la simulation du graphe de HMM. Dans notre cas, les événements correspondent aux séquences de durées $\{d_i\}$. Lorsqu'un symbole ∞ est émis par un état du HMM, celui-ci reste en attente d'un événement à l'image d'un modèle atomique DEVS. Si un événement apparaît, il est traité et le HMM sort de son état "infini". Au contraire si un événement apparaît alors qu'aucun symbole ∞ n'a été émis, celui-ci est ignoré. La prise en compte de ces informations a nécessité la modification de l'algorithme de génération de séquences. Contrairement au générateur de séquences d'un HMM conventionnel, celui-ci est piloté par la présence du symbole ∞ et l'apparition d'événements. L'algorithme 24 présente les modifications apportées au générateur de séquences d'observations pour tendre vers un modèle DEVS classique.

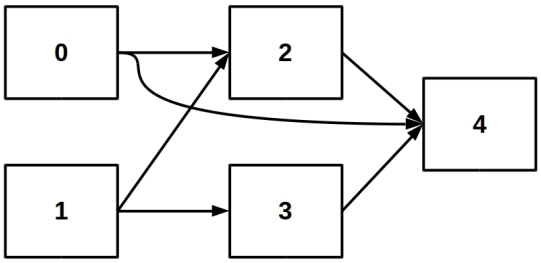


FIGURE 3.8 – Exemple de graphe de HMM

Le déroulement de la simulation du graphe de HMM s'effectue en suivant la structure du graphe. Prenons l'exemple de la figure 3.8, pour que le modèle 2 soit simulé il est nécessaire de connaître les entrées auxquelles il est soumis. Ces entrées sont elles-mêmes générées par les HMM 0 et 1. La simulation du HMM 2 aura forcement lieu après la simulation des HMM 0 et 1. Cette relation de dépendance nécessite le respect d'une certaine logique lors de l'exécution des algorithmes de génération de séquences dans le graphe de HMM. Pour cela, il est nécessaire de restructurer le graphe de HMM de façon à garantir le respect de l'ordre des générations de séquences. Le non respect de cet ordre provoquerait à coup sûr des erreurs lors de la génération de séquences, ce qui fausserait totalement la pondération du graphe de modèle. La restructuration du graphe consiste à disposer les sommets en couche, ce qui garantit que tout HMM d'une couche est exécutable à condition que tous les HMM de la couche précédente ont été simulés. Cette restructuration garantit le bon déroulement de la simulation du graphe de modèle.

Algorithme 24 Algorithme de génération de séquences d'observations pour un HMM représentant un modèle avec entrée

PROCÉDURE: Générateur adapté aux modèles avec entrées
ENTRÉES: $\lambda = (\pi, A, B)$, D dates d'arrivées des événements externes, t_{max} temps d'arrêt
  $I = 0$ est l'index de la date courante contenue dans D
  Tirage d'un état initial S_i suivant les probabilités π
  $S_{courant} \leftarrow S_i$ devient l'état courant
  Tirage d'un symbole $v_k \in V$ suivant les probabilités $b_{i,k}$
  $seq \leftarrow v_k$ ajout du symbole tiré à la séquence d'observations
  $t_{last} = 0$ temps durant lequel le HMM est en attente
  Tantque $t < t_{max}$ **faire**
    $t = 0$
    Tirage d'un état futur S_j suivant les probabilités a_{ij}
    $S_{courant} \leftarrow S_j$ devient l'état courant
    Si $S_{courant} = S_{final}$ **Alors**
      Tirage d'un nouvel état initial S_j suivant les probabilités π
      $S_{courant} \leftarrow S_j$ devient l'état courant
    Finsi
    Tirage d'un symbole v_k suivant les probabilités b_{ij}
    Si $v_k \neq \infty$ **Alors**
      $seq \leftarrow seq \cup \{t - t_{last} + v_k\}$ ajout du symbole $\{v_k +$ le temps passé en attente $\}$ à seq
      $t_{last} \leftarrow t$
      $t \leftarrow t + v_k$
      Tantque $D_I \leq t$ **faire**
        $I \leftarrow I + 1$
      Fin Tantque
    Sinon
      $t \leftarrow D_I$
      $I \leftarrow I + 1$
    Finsi
  Fin Tantque
  Sorties: seq

L'algorithme 25 présente la procédure sur laquelle repose la restructuration du graphe de

HMM. Celui-ci commence en ré-indexant tous les sommets ne possédant pas d'arcs entrants. Ces sommets forment la première couche de la nouvelle structure du graphe. Ensuite, pour chaque arc sortant de la première couche, on étudie le sommet auquel il est rattaché. Si ce sommet est uniquement dépendant d'un sommet d'une des couches précédentes alors il est ré-indexé et attribué à la couche suivante. Au contraire, si ce sommet possède des arcs entrants provenant d'autres sommets, il n'est pas modifié tant que les sommets dont il dépend ne l'ont pas été. Ce processus est réitéré tant que tous les sommets du graphe n'ont pas été ré-indexés.

Algorithme 25 Algorithme de ré-indexation des sommets du graphe en couche pour la génération de séquence des HMM

PROCÉDURE: Ré-indexation en couche
ENTRÉES: $G(V,E)$
  $G_{copie} = G$ copie du graphe
  $Id = 0$ indice de réindexation
  Pour chaque v dans V **faire**
    Si v ne possède pas d'arc entrant **Alors**
      l'indice du sommet v dans V_{copie} devient Id
      verrouillage du sommet v dans V
      $Id = Id + 1$
    Finsi
  Fin pour
  Tantque des sommets v sont disponibles **faire**
    Pour chaque v dans V non verrouillé **faire**
      Si v possède des arcs entrants **Alors**
        recherche des sommets $\{v_i\} \in V$ aux extrémités des arcs entrants
        Si tous les $\{v_i\} \in V$ sont verrouillés **Alors**
          l'indice du sommet v dans V_{copie} devient Id
          verrouillage du sommet v dans V
          $Id = Id + 1$
        Finsi
      Finsi
    Fin pour
  Fin Tantque
  $G = G_{copie}$
  Sorties: G

3.4 Validation des approches d'apprentissage de la dynamique des modèles atomiques

La qualité de l'apprentissage de la dynamique d'un modèle DEVS dépend en grande partie de la qualité de la méthode utilisée, ainsi que de l'implémentation de l'algorithme d'apprentissage. Nous avons vu dans les sections précédentes que ce seul algorithme ne suffit pas à garantir l'obtention d'un apprentissage de qualité. En effet, la génération d'une base d'apprentissage reflétant au mieux la réalité du modèle est primordiale. Comment peut-on évaluer la qualité d'un apprentissage ? Quels facteurs peuvent influencer cette qualité ? Le rôle de cette sous-section est

de présenter une approche pour la validation de notre méthode d'apprentissage. Cette validation servira comme base de comparaison aux futurs apprentissages réalisés dans cette section. Toutes ces stratégies nécessitent la création d'un benchmark de graphes artificiels à partir duquel les apprentissages, ainsi que les simulations seront réalisées.

3.4.1 Présentation des données

La réalisation d'un plan d'expérience, tant pour l'apprentissage que pour la simulation distribuée, a nécessité la mise en place de graphes de modèles atomiques auxquels sont associés des dynamiques artificielles. Le partitionnement, ainsi que l'apprentissage sont tous deux dépendants de la structure du graphe de modèles. C'est pourquoi nous avons créé plusieurs catégories de générateur de graphes, dont la structure et les connections varient. Différents tests sont réalisés sur chaque catégorie de graphe afin de réaliser une expérimentation variée, permettant une généralisation des résultats obtenus. Le rôle de cette sous-section est de présenter chaque catégorie de graphes, ainsi que la dynamique des modèles atomiques qui les composent.

Les graphes en forme de grille

La première catégorie de graphes sur laquelle nous avons travaillé est celle en forme de grille. Ces graphes portent le nom de ***grid-graph***, car leur structure sous forme de maillage leur donne l'aspect d'une grille. Chaque graphe se définit par un nombre de sommets de la forme $n \times n$, où n correspond au nombre de sommets qui est contenu sur une largeur de grille. En outre, cette grille peut être représentée géométriquement par un losange de coté n, comme le montre l'exemple de la figure 3.9, où $n = 5$.

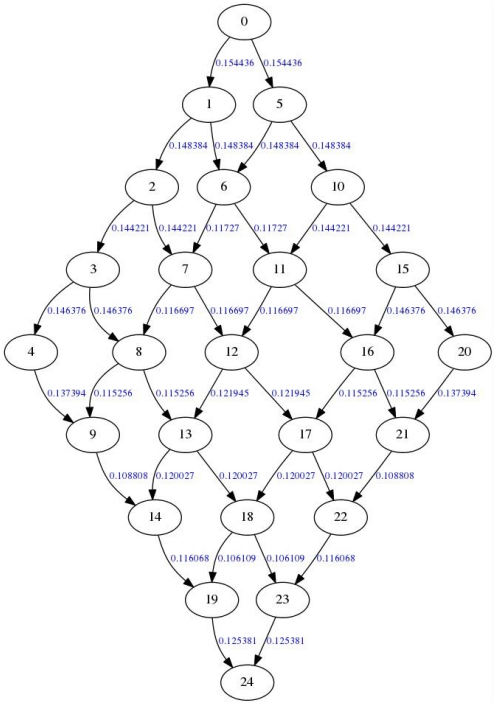


FIGURE 3.9 – Exemple de ***grid-graph*** de taille 25 pondéré

Quel que soit le nombre de sommets, la structure du graphe reste toujours la même : partant d'un sommet sans entrée à l'extrémité supérieure de la grille, chaque sommet possède une connexion avec chacun des deux voisins les plus proches se situant sur le niveau inférieur. Seul le dernier sommet du graphe ne possède pas d'arc sortant. Si l'on considère un tel graphe non-orienté, le nombre de voisins que peut posséder un sommet est donc compris entre 2 et 4. Les sommets aux quatre extrémités possèdent deux voisins, les autres situés sur le bord en possèdent trois et tous ceux à l'intérieur du graphe en ont quatre.

L'intérêt de cette catégorie de graphe réside dans sa structure simple. En effet, le faible nombre de connexions, ainsi que sa structure répétitive en font un exemple simple à étudier. Une observation du graphe permet d'estimer d'une part la qualité du partitionnement et d'autre part celle d'un apprentissage, ce qui en fait un très bon candidat pour la validation d'algorithme. En section 2.2.2, nous parlions de deux catégories de structure de graphe : l'une dite "simple" avec un nombre de connexions faible et l'autre dite "complexe" avec un nombre de connexions important. Ce type de graphe entre dans la catégorie des graphes "simples". Cette distinction est importante, car nous verrons qu'en fonction de la structure du graphes les résultats d'apprentissage peuvent fortement varier.

Les graphes en forme "d'arbre"

Cette catégorie de graphe porte abusivement le nom de ***tree-graph***, car leur forme fait penser aux arbres de probabilité comme l'illustre parfaitement la figure 3.10. En effet, les graphes en forme d'arbre existent dans la littérature, cependant dans le cadre de cette thèse, nous considérerons les ***tree-graph*** comme étant ceux présentés dans cette section. La création de ces graphes est inspirée par la modélisation des écoulements d'eau dans les bassins versants. Afin de représenter au mieux ce genre de situation, les graphes générés sont orientés et sans cycle. Ils se composent d'un ensemble de sommets appelés *sources* qui sont des sommets sans arc entrant. À ces sommets sont rattachés d'autres sommets formant les branches de l'arbre qui sont elles-mêmes rattachées les unes aux autres jusqu'à atteindre l'unique sommet sans arc sortant appelé *exutoire*.

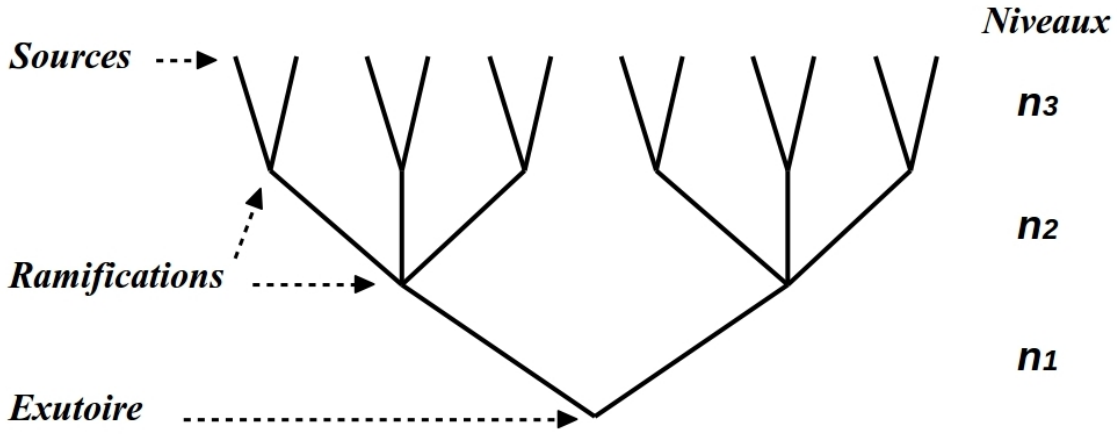


FIGURE 3.10 – Illustration de la structure d'un ***tree-graph*** de niveau $\{2,3,2\}$

Ces graphes ont une structure particulière, ils se composent de α branches, où chaque branche contient le même nombre de sommets interconnectés. La détermination du nombre de branches α se fait par le biais du vecteur $niveau = \{n_1, n_2, \cdots, n_k\}$, où chaque n_i correspond au nombre de branches du niveau i. Ce vecteur donne donc le nombre de branches qui doivent être ramifiées à chacune des branches du niveau inférieur. L'évolution des niveaux se faisant de bas en haut, on retrouvera donc le niveau n_0 en bas du graphe et n_k en haut du graphe. Dans l'exemple de la figure 3.10, $niveau = \{2,3,2\}$ c'est à dire que le graphe se compose de 2 branches au niveau 0, où chacune des ces branches se ramifie à 3 autres branches formant un total de 6 branches au niveau 1 et ainsi de suite. Le nombre de branches d'un ***tree-graph*** et le nombre de sommets par branche s'obtiennent comme suit

$$\alpha = \sum_{i=1}^{k} \prod_{j=1}^{i} n_j \quad \text{et le nombre de sommets par branche } = \frac{\text{nombre de sommets}}{\text{nombre de branches}}$$

Chaque branche du graphe est connectée à d'autres branches au niveau d'une *ramification*. Une ramification est en réalité la connexion de plusieurs sommets appartenant aux extrémités de différentes branches. Elle a pour but de lier les branches entre elles et de donner sa forme au graphe. Toutes les branches sont ramifiées en fonction de leur niveau jusqu'à atteindre l'exutoire. Chaque branche se compose d'un même nombre de sommets, où chaque sommet est connecté à 2 ou 3 sommets appartenant à la même branche suivant une logique spatiale descendante. Cette politique de connexion permet d'obtenir des sommets dont le nombre de voisins peut être compris entre 2 et 9.

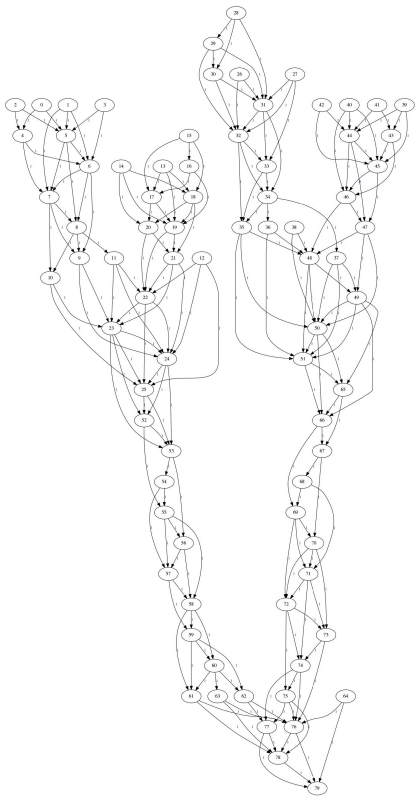


FIGURE 3.11 – Exemple de ***tree-graph*** de taille 80 et de niveau $\{2,2\}$

L'intérêt de cette catégorie de graphes est qu'elle offre un certain niveau de complexité dans ses connexions et possède cependant une structure relativement simple. Contrairement au ***grid-graph***, la génération d'un de ces graphes repose sur un processus aléatoire de sélection de voisins. Ce qui implique que pour deux générations distinctes avec les mêmes paramètres initiaux, les graphes résultants seront différents. La variabilité des paramètres offre une flexibilité que n'ont pas les ***grid-graph***. L'augmentation de la taille du vecteur *niveau* permet d'augmenter la profondeur du graphe et l'augmentation des $n_i \in niveau$ permet d'augmenter sa largeur. Il est également possible de modifier le nombre moyen de voisins par branche pour complexifier la structure. L'intérêt de ces graphes pour l'apprentissage est principalement la taille des chaînes formées par la connexion des sommets. En effet, dans ce cas les chaînes peuvent atteindre des tailles très importantes, comme le montre le graphe de taille 80 de la figure 3.11 se composant de chaînes pouvant atteindre jusqu'à 26 sommets. Grâce à ces graphes, nous allons pouvoir étudier l'influence de la taille des chaînes sur la qualité d'apprentissage.

Les graphes à structure hyper connectée

La création de cette catégorie de graphes reprend l'idée du bassin versant mais en complexifiant d'avantage les connexions entre les sommets. L'idée étant de créer un graphe orienté sans cycle fortement connecté, portant le nom de ***linked-graph***. Celui-ci dispose d'une structure en couches, où chaque sommet d'une couche est connecté à un ou plusieurs sommets de la couche inférieure. Partant d'une première couche ne contenant que des sommets ***sources*** (sans arc entrant), la connexion des sommets s'effectue de couche en couche jusqu'à atteindre l'exutoire (l'unique sommet sans arc sortant). Comme le montre la figure 3.12, ce type de graphe possède une structure triangulaire où les connexions entre sommets y sont très denses.

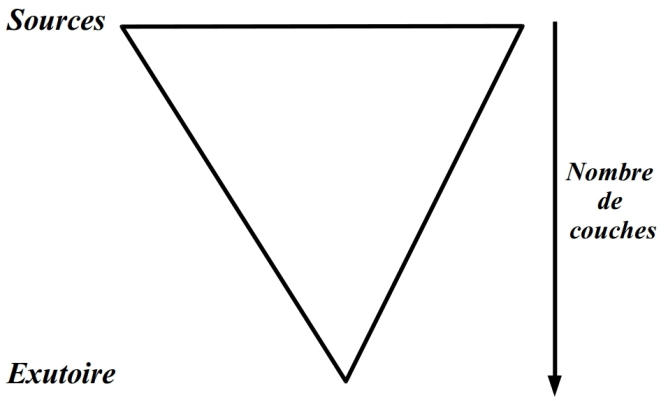


FIGURE 3.12 – Illustration de la structure d'un ***linked-graph***

Partant d'un nombre de sommets n, la création d'un ***linked-graph*** à k couches dépend du nombre de sommets contenus dans la première couche. En effet, il est possible d'attribuer $\frac{n}{\alpha}$ sommets à la première couche, où α est choisi par l'utilisateur. Quel que soit le paramétrage de α et de n, la dernière couche ne contient qu'un unique sommet. À partir de ces informations, il est possible de connaître le nombre de sommets à répartir au sein des $k-2$ couches intermédiaires : $nbrSC = n - \frac{n}{\alpha} - 1$. Le nombre de sommets varie en fonction de l'indice de la couche dans lequel

ils se situent. Le nombre de sommets de la couche i, noté $nbrSC_i$, s'obtient par :

$$\beta = \sum_{i=2}^{k-1} i$$
$$nbrSC_i = \frac{i \times nbrSC}{\beta}$$

Chaque sommet d'une couche i est connecté avec deux ou trois sommets de la couche $i-1$, sauf les sommets de la couche 2 qui n'admettent qu'un unique arc les reliant à l'exutoire. La figure 3.13 montre un exemple de ***linked-graph*** à 150 sommets répartis sur 12 couches, où la première couche contient $\frac{1}{5}$ des sommets du graphe.

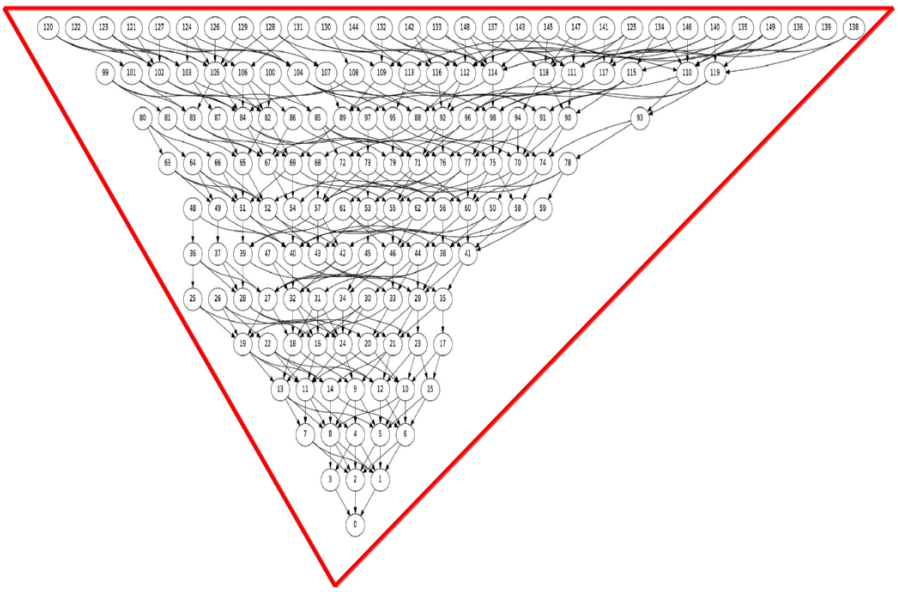


FIGURE 3.13 – Exemple de ***linked-graph*** à 12 couches

Ce type de graphe entre dans la catégorie des graphes "complexes" de part leur forte densité de connexions. Leur intérêt réside dans cette complexité, car il est évident qu'il est bien plus difficile d'apprendre ou de partitionner de tels graphes. Cependant, ils possèdent un grand nombre de modèles sans entrée dont la dynamique est facile à apprendre. Quel sera l'impact de cette caractéristique sur la qualité de l'apprentissage ? Nous tâcherons de répondre à cette question en section 3.4.3.

Les graphes à structure dérivée de celle de DEVStone

DEVStone [GW05], créé par Ezequiel Glinsky et Gabriel Wainer, est un générateur de modèles synthétiques dont le rôle est de produire différentes variétés de modèles possédant des structures et des interactions diverses, tout en effectuant un mélange d'opérations communes. La production de modèles via DEVStone repose sur l'utilisation de différents paramètres tels que : le type (variation

de la structure et des connexions entre les composants), la profondeur d (nombre de niveaux dans la hiérarchie de modèles) et la largeur w (nombre de composants dans chaque modèle couplé intermédiaire). Il existe trois types de modèles :

LI : modèle possédant un faible niveau d'interconnexions

HI : modèle possédant un fort niveau de couplages en entrée

HO : modèle possédant de nombreuses sorties

Dans le cadre de cette thèse, nous nous intéressons au graphe de modèles que pourrait fournir la mise à plat d'un modèle issu de DEVStone. Plus précisément, sur les modèles DEVStone de type **HI**. Un tel modèle se compose de d modèles couplés où chacun d'entre eux se compose de $w-1$ modèles atomiques à l'exception du dernier niveau qui se compose d'un unique modèle atomique. Pour chaque niveau, le modèle atomique i est connecté au modèle atomique $i+1$ pour tout $i < w-1$. La mise à plat d'un modèle **HI** donne naissance à un graphe de modèles totalement dépendant des paramètres $\{d,w\}$, se composant de $(d-1)*(w-1)+2$ modèles atomiques. La figure 3.14 montre un exemple de structure hiérarchique d'un modèle **HI** de paramètre $\{3,3\}$, et le graphe de modèle obtenu par mise à plat de cette hiérarchie.

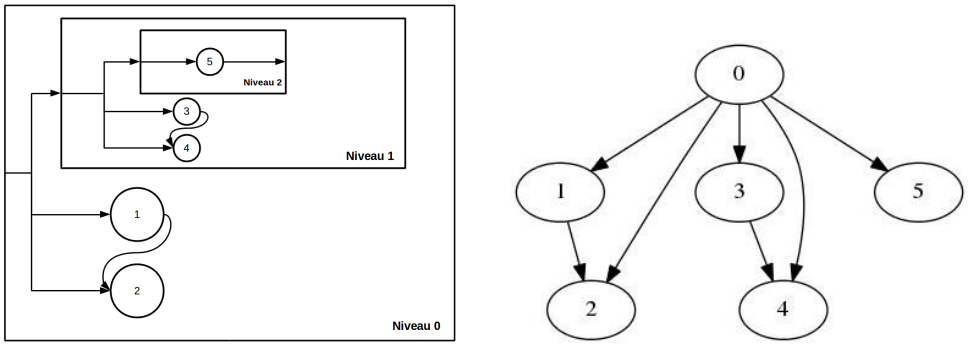


FIGURE 3.14 – Structure hiérarchique d'un modèle DEVStone HI standard de paramètres (3,3) et son graphe de modèles atomiques issu de la mise à plat

En plus d'être un exemple standard de la littérature, un modèle DEVStone de type **HI** offre une flexibilité dans son paramétrage qui permet d'étudier l'impact de la forme d'un graphe sur la qualité de l'apprentissage. Cependant, la structure du graphe d'un modèle **HI** est relativement trop "simple" pour observer l'impact de la dépendance des modèles sur l'apprentissage, ainsi que pour être utilisé pour une distribution. C'est pourquoi nous proposons une légère modification de ce type de modèles que nous appelons DEVStone **HIM**, pour **HI modifié**.

Notre modèle DEVStone **HIM** se définit par 3 paramètres : d la profondeur, w la largeur des modèles couplés de type A et v la profondeur des modèles couplés de type B. En effet, contrairement au modèle **HI** standard, les **HIM** distinguent deux types de modèles couplés dont les structures et le nombre de modèles qui les composent diffèrent. Sa structure hiérarchique, illustrée en figure 3.15, se compose au total de $\sum_{i=1}^{d} i$ modèles couplés où chaque modèle de type A contient $w-1$ modèles atomiques et chaque modèle couplé de type B contient v modèles atomiques. Pour les modèles couplés de type A, la connexion entre les modèles atomiques d'un même niveau est le même que pour un DEVStone **HI**. Cependant, si le niveau courant est différent du dernier niveau

de la hiérarchie, alors toute sortie d'un modèle atomique i est connectée à l'entrée d'un modèle couplé de type B à condition que $i \neq w-1$. Pour ce qui est des modèles couplés de type B, le modèle atomique j est connecté au modèle atomique $j+1 \ \forall j \in [1, w-1[$. Cependant, si le niveau courant est différent du dernier niveau alors la sortie du modèle atomique $w-1$ est connectée à l'entrée d'un modèle couplé de type B.

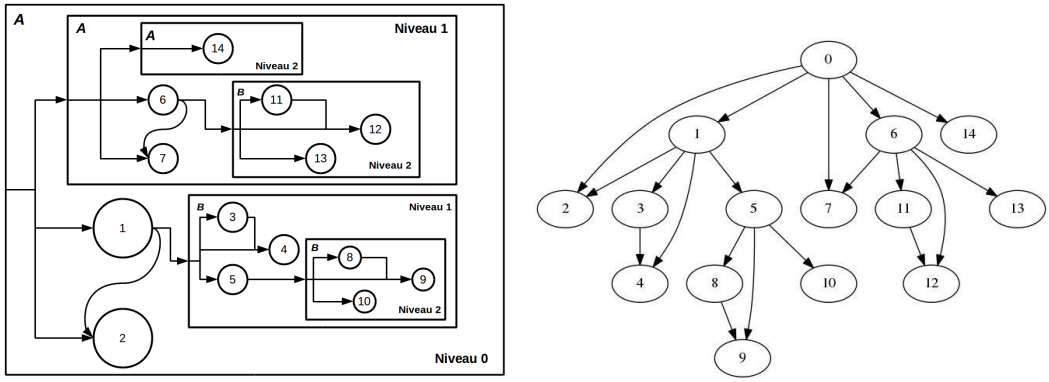


FIGURE 3.15 – Structure hiérarchique d'un modèle DEVStone HIM de paramètres (3,3,3) et son graphe de modèles atomiques issu de la mise à plat

Pour réaliser une simulation, la sortie d'un modèle atomique est mise en entrée d'un modèle DEVStone **HIM**. Le graphe issu de la mise à plat de la hiérarchie d'un tel modèle est illustré à droite de la figure 3.15. Le graphe d'un modèle DEVStone **HIM** de paramètres $\{d, w, v\}$ se compose de $(d-1)*(w-1)+2+\sum_{i=1}^{d-1}(w-2)*v*(d-i)$ modèles atomiques. Contrairement à un **HI** standard, le **HIM** offre un graphe de modèle évoluant en profondeur, en largeur mais aussi en longueur. De plus, même s'il reste dans la catégorie des graphes "simples", il a l'avantage d'offrir des structures répétitives permettant la formation de longues chaînes.

Exemple de dynamique : remplissage de réservoirs

La validation de nos méthodes d'apprentissage a nécessité la création d'une dynamique artificielle commune à chaque modèle atomique. Celle-ci simule le remplissage de cinq réservoirs de même contenance initialement vides, où chaque réservoir possède sa propre vitesse de remplissage v_i. La dynamique est la suivante : le système se trouve dans un état initial (parmi les états disponibles) tant qu'aucun réservoir n'est rempli, c'est à dire tant qu'aucun réservoir n'a atteint sa capacité maximale Q_{max}. Lorsqu'un réservoir atteint sa valeur Q_{max}, la fonction λ est appelée pour effectuer une sortie correspondant : soit à une quantité, soit à une vitesse, en fonction de la dynamique souhaitée. Le remplissage du réservoir est ensuite stoppé et la fonction δ_{int} est appelée pour changer d'état. Lorsque K réservoirs atteignent leur capacité maximale, ils sont vidés et peuvent de nouveau être remplis. Dans le cadre de l'apprentissage, une durée d_i est émise à chaque fois qu'un réservoir est rempli. La figure 3.16 illustre cette dynamique et donne un exemple d'automate à états finis représentant la dynamique du système.

Le réseau de connexions entre les modèles atomiques correspond aux échanges entre les réservoirs appartenant à différents modèles. Si un modèle A émet une vitesse V_{emi} à un modèle B,

celui-ci verra la vitesse de l'un de ses réservoirs changer. Dans ce cas, la dynamique des modèles atomiques recevant des événements se verra modifiée. En effet, si l'on prend l'exemple de la figure 3.16, imaginons que ce modèle voit la vitesse de remplissage de son cinquième réservoir modifiée ($V_5 < V_{emi}$). Dans ce cas, le réservoir R_5 aura terminé son remplissage avant le réservoir R_1. Or, la dynamique initiale aurait voulu que R_1 soit rempli en premier, alors que dans ce cas c'est R_5 qui est rempli, ce qui a pour conséquence de modifier la séquence d'états du modèle et la génération des d_i. Cependant, si c'est une quantité de matière, celle-ci n'est qu'une information et par conséquent, elle ne modifiera en rien la dynamique du modèle récepteur. Cette réception d'événements fait appel à la fonction de transition externe, qui a pour rôle de modifier l'état courant et de choisir aléatoirement le réservoir qui recevra la nouvelle vitesse, ou l'information concernant la quantité de matière.

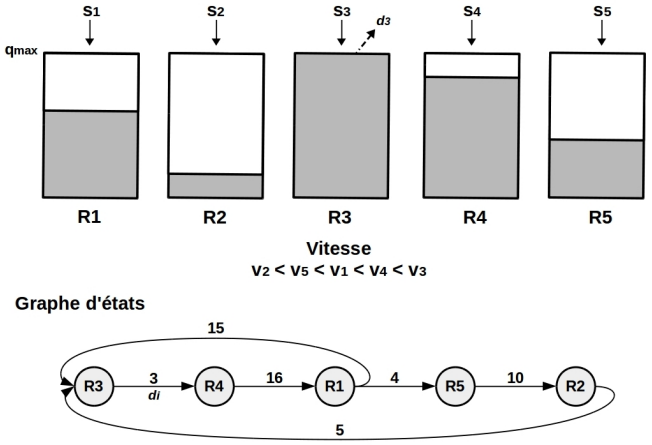


FIGURE 3.16 – Illustration de la dynamique de remplissage pour cinq réservoirs

Cet exemple de dynamique a un double avantage : dans un cas elle permet de montrer l'impact que peuvent avoir les entrées d'un modèle sur sa dynamique initiale (pour le changement de vitesse) et dans l'autre cas, elle permet d'avoir des échanges d'événements sans pour autant modifier la dynamique. C'est pourquoi nous avons fait le choix d'inclure l'une de ces dynamiques dans chacun de nos modèles atomiques pour nos différentes phases de tests, tant pour l'apprentissage que la simulation.

3.4.2 Validation de l'approche d'apprentissage minimaliste

Le rôle de cette sous-section est d'estimer, en fonction de la procédure d'apprentissage suivie, la meilleure pondération que l'on peut espérer en utilisant les véritables observations, obtenues par simulation du modèle DEVS, comme base d'apprentissage. Deux procédures sont comparées : l'une consiste à apprendre la dynamique de chaque modèle atomique (méthode totale) et l'autre repose sur un apprentissage minimaliste des dynamiques qui composent le modèle DEVS. On entend par minimaliste, le fait qu'on réalise un unique apprentissage par dynamique et par nombre d'entrées que contiennent les modèles atomiques. Il est évident que la première approche est utopique, au sens où le temps nécessaire à l'apprentissage de centaines de milliers de modèles serait plus pénalisant que le gain apporté par la pondération des arcs du graphe de modèles pour le par-

titionnement. Cependant, elle est nécessaire pour servir de base de comparaison afin d'estimer la qualité réelle de notre approche par apprentissage minimaliste. Les résultats obtenus par celle-ci sont, en quelque sorte, une borne supérieure vers laquelle nous cherchons à tendre.

Afin d'estimer la qualité de notre méthode d'apprentissage, nous allons comparer chaque poids obtenus à partir d'une séquence d'observation, résultant de la simulation du graphe de HMM, à ceux obtenus à partir de la séquence d'observations résultant de la simulation du modèle DEVS. Le but étant d'évaluer l'erreur moyenne d'apprentissage optimale attendue pour chacune des catégories de graphes précédemment présentées. Nous parlons ici d'erreur relative exprimée en pourcentage par le biais de la formule

$$\Phi(v_i) = \frac{\mid poids_{\text{estimé}}(v_i) - poids_{\text{réel}}(v_i) \mid}{poids_{\text{réel}}(v_i)}$$

Les graphes de modèles atomiques, présentés en section 3.4.1, issus de la mise à plat des quatre catégories de modèles DEVS ont tous une taille d'environ 300 et se composent de modèles atomiques dont la dynamique correspond à celle du remplissage de réservoirs présentée ci-dessus. Par souci de clarté et de rapidité, les dynamiques ne sont pas soumises à de longs calculs dans ce cas. La séquence d'observations est obtenue par simulation du modèle DEVS durant dix secondes.

Validation pour chaque catégorie de graphes

Cette sous-section présente une étude des erreurs d'apprentissage pour chaque catégorie de graphe et pour chaque procédure d'apprentissage, à l'aide de boîtes à moustaches. En effet, cette représentation des erreurs a l'avantage de fournir d'importantes informations statistiques, tout en offrant un aspect visuel très intéressant. Nous n'avons pas opté pour une représentation graphique des erreurs modèle atomique par modèle atomique, car en vue de la taille des graphes utilisés pour cette validation le résultat n'en serait qu'illisible.

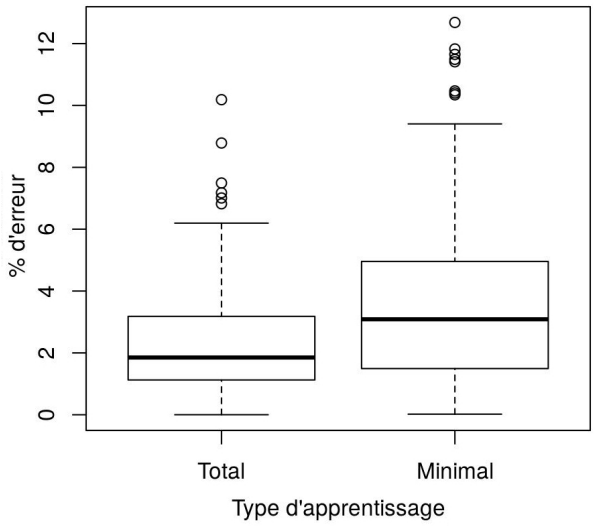


FIGURE 3.17 – Boxplots des erreurs d'apprentissage d'un grid-graph, pour la méthode totale à gauche et la méthode minimaliste à droite

Commençons par étudier les erreurs d'apprentissage pour chacune des procédures sur un *grid-graph*. Rappelons que ce graphe se compose de 288 modèles atomiques, et a par conséquent nécessité 288 apprentissages distincts pour obtenir les résultats pour la procédure totale. Dans ce cas, le temps nécessaire pour y parvenir fut d'environ 40 minutes. Or, l'apprentissage par la procédure minimaliste n'a nécessité que 3 apprentissages distincts et par conséquent un temps inférieur à 22 secondes. En effet, le nombre d'entrées de chaque modèle étant compris entre 0 et 2, le nombre d'apprentissages s'en voit considérablement réduit. Toutefois ces temps sont arbitraires et totalement dépendants de la taille de la séquence d'apprentissage qui dépend elle-même de la dynamique du modèle simulé. Pour un même laps de temps, deux modèles dont la dynamique diffère pourront fournir des séquences d'apprentissage de tailles différentes. Ces temps sont donc à prendre avec un certain recul. Intéressons nous maintenant aux erreurs produites par chaque procédure d'apprentissage, présentées par la figure 3.17. Celle-ci nous informe que les erreurs maximales observées sur ces deux procédures sont inférieures à 13%. De plus, la procédure par apprentissage total possède une erreur moyenne d'environ 2%, contre 3% pour la procédure minimaliste. Même si la répartition des erreurs est légèrement meilleure pour la procédure totale que pour la minimaliste, cette dernière offre des résultats tout à fait satisfaisants pour un temps d'apprentissage nettement plus faible. Au vu de ces résultats, il semblerait que la procédure par apprentissage minimaliste supporte bien l'apprentissage des dynamiques qui compose les graphes "simples". En effet, pour cette catégorie de graphe un modèle peut contenir au maximum deux entrées. Confirmons cette hypothèse par l'étude d'un graphe "complexe".

La même démarche est appliquée pour les 300 modèles d'un *tree-graph* de paramètre $\{3, 2, 3\}$. Contrairement au *grid-graph*, les modèles qui composent ce graphe contiennent entre 0 et 10 entrées, soit cinq fois plus que pour le précédent graphe. La figure 3.18 présente les erreurs d'apprentissage obtenues par application des deux procédures sur ce type de graphe.

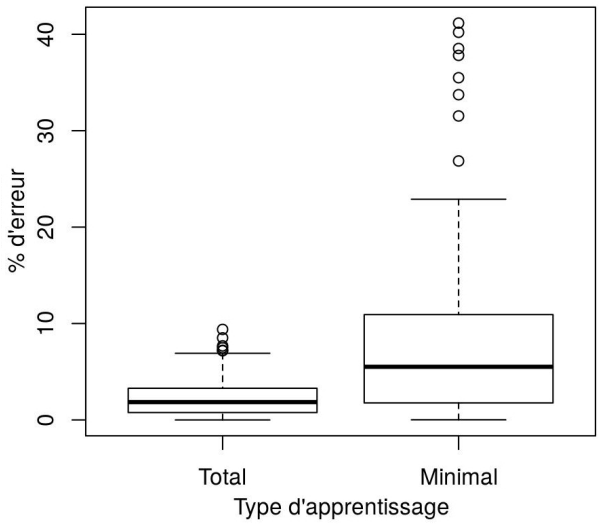


FIGURE 3.18 – Boxplots des erreurs d'apprentissage d'un tree-graph, pour la méthode totale à gauche et la méthode minimaliste à droite

Ces boxplots montrent que, dans ce cas, la procédure par apprentissage totale offre de meilleurs résultats que la procédure minimaliste. En effet, l'erreur maximale de la procédure totale est de

10% contre 42% pour la minimaliste. Cependant, en observant d'un peu plus près les boxplots on peut constater qu'en réalité les erreurs supérieures à 20% sont faibles. De plus, l'erreur moyenne de la procédure totale est d'environ 2% contre environ 7% pour la procédure minimaliste. Cet écart peut s'expliquer par deux hypothèses : la première est liée au nombre d'entrées des modèles et la seconde à la taille des chaînes formées par ces mêmes modèles atomiques. En effet, en supposant qu'une chaîne se compose de N HMM comme l'illustre la figure 3.19 et que la moyenne des erreurs d'apprentissage des modèles varie entre 2 et 6 pour-cents, l'application d'un algorithme de génération de séquences peut expliquer l'apparition d'une erreur maximale en bout de chaîne. En effet, la génération de la séquence du HMM $i+1$ dépendant de la séquence générée par le HMM i qui dépend lui-même des séquences générées par les HMM $j < i$. Il y a donc une rétro-propagation de l'erreur lors de la génération de séquences sur chaque HMM qui compose la chaîne, ce qui a tendance à l'amplifier.

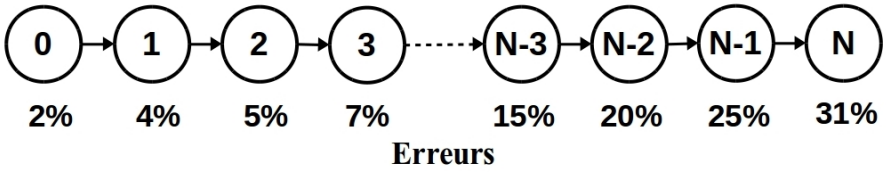


FIGURE 3.19 – Exemple de chaînes de HMM et impact sur l'erreur d'apprentissage

Étudions maintenant l'apprentissage des dynamiques des modèles d'un *linked-graph* de taille 300 structuré en 15 couches. Contrairement au *tree-graph*, cette catégorie de graphe possède un réseau de connexions nettement plus dense, un nombre de modèles sans entrée plus important et le nombre d'entrées par modèle est compris entre 0 et 26. Au vu de ce paramétrage, il semble légitime de penser que l'apprentissage des dynamiques d'un tel graphe soit plus complexe que celui d'un *tree-graph* et nettement plus difficile que celui d'un *grid-graph*. Tentons de confirmer cette hypothèse par l'étude des erreurs d'apprentissage contenues dans la figure 3.20.

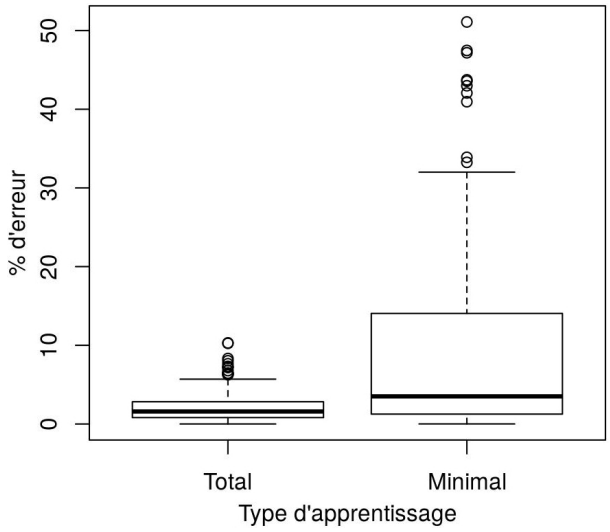


FIGURE 3.20 – Boxplots des erreurs d'apprentissage d'un linked-graph, pour la méthode totale à gauche et la méthode minimaliste à droite

Comme pour les autres types de graphes, la procédure d'apprentissage total offre une erreur maximale aux alentours de 10%, pour une moyenne d'environ 2%. La procédure minimaliste offre une erreur maximale d'environ 50%, pour une erreur moyenne d'environ 9%. En observant d'un peu plus près la figure 3.20, on peut observer que seulement $1/4$ des erreurs sont supérieures à 17%. Il est tout de même important de rappeler que ce type de graphe se compose de 100 modèles sans entrée, qui sont ceux qui offrent les meilleures erreurs d'apprentissage ($< 3\%$). En effet, les modèles de la figure 3.21 sont triés par ordre croissant de leur nombre d'entrées, ainsi les modèles proches de l'origine du repère n'ont aucune entrée et ceux proches de l'autre extrémité ont 26 entrées. Le nombre de modèles comportant beaucoup d'entrées reste cependant faible (moins d'un pour-cent), par contre le nombre de modèles sans entrée représente $1/3$ des modèles atomiques du graphe (soit 100 modèles). Cette représentation de l'évolution de l'erreur d'apprentissage permet d'observer l'impact qu'a le nombre d'entrées d'un modèle atomique sur sa qualité d'apprentissage. Nous pouvons donc émettre l'hypothèse qu'un graphe possédant une structure similaire mais ne contenant qu'un dixième de modèles sans entrée verrait son erreur d'apprentissage moyenne augmenter aux alentours de 20%. Cette remarque tend à confirmer les hypothèses émises lors de l'étude des résultats de l'apprentissage du *tree-graph*. En effet, il semblerait que plus le nombre d'entrées d'un modèle est important, plus la procédure minimaliste rencontrerait des difficultés lors de la génération de séquences d'observation.

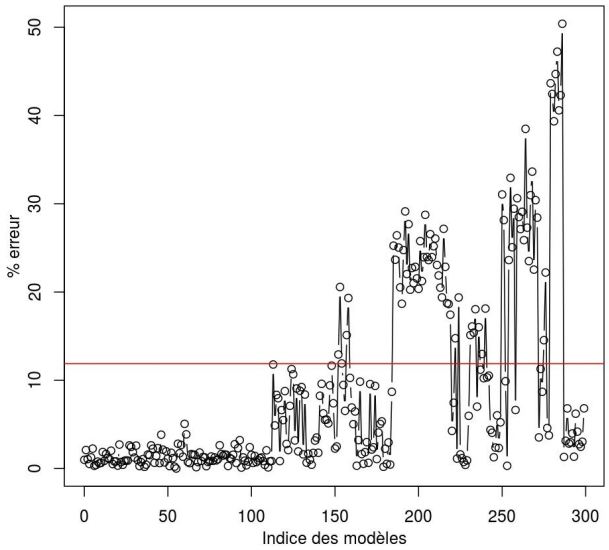


FIGURE 3.21 – Évolution de l'erreur d'apprentissage en fonction de la localisation du modèle dans le graphe et de son nombre d'entrées

En observant d'un peu plus près la figure 3.21, on peut constater que les modèles comportant plus de 20 entrées offrent de meilleurs résultats que ceux qui en comportent moins. Comment expliquer ces résultats ? Après une observation détaillée du graphe sur lequel est réalisé cet apprentissage, nous avons pu constater que les modèles possédant un très grand nombre d'entrées se trouvent sur la deuxième couche de sommets du graphe. Ce qui signifie qu'ils sont obligatoirement les deuxièmes maillons d'une chaîne de modèles. Or, en suivant l'hypothèse émise précédemment sur l'évolution de l'erreur d'apprentissage au sein d'une chaîne, les séquences mises en entrée de ces

modèles comportent une faible erreur qui ne pénalise pas la génération de séquence d'observation par simulation des HMM, contrairement à un modèle en bout de chaîne. La qualité de l'apprentissage d'un modèle est donc dépendant, à la fois de son nombre d'entrées et de sa position au sein d'une chaîne.

Confirmons cette dernière hypothèse à l'aide d'un graphe "simple" de type *DEVStone_HIM* de taille 327 et de paramètre $\{6,6,5\}$. Dans ce graphe, les modèles contiennent entre 0 et 2 entrées, à l'image du *grid-graph*. DEVStone étant un outil utilisé et approuvé par la communauté DEVS, nous avons fait le choix de présenter les résultats d'apprentissage de ce type de graphe même si celui-ci offre des résultats proches de ceux des *grid-graph*. La figure 3.22 présente les erreurs d'apprentissages obtenues pour chaque procédure sur cette catégorie de graphes.

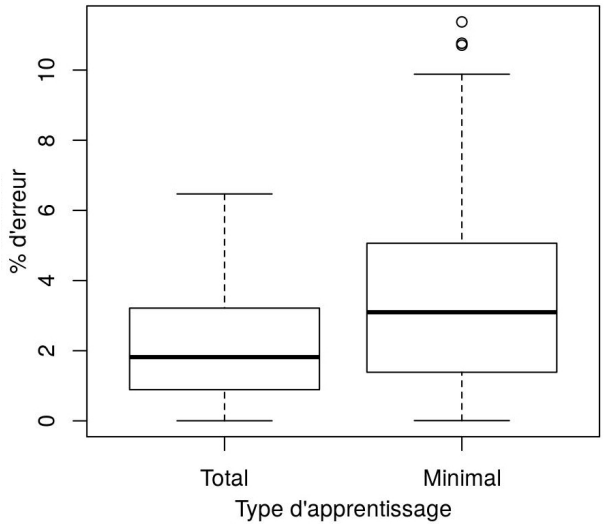


FIGURE 3.22 – Boxplots des erreurs d'apprentissage d'un DEVStone_HIM-graph, pour la méthode totale à gauche et la méthode minimaliste à droite

Nous pouvons constater que les résultats obtenus par application des deux procédures sont relativement proches dans ce cas. En effet, si l'on compare les erreurs maximales, nous pouvons constater que la procédure totale offre une erreur de 7% contre une erreur de 11% pour la procédure minimaliste. De plus, ces deux procédures ont une erreur moyenne relativement proche entre 2 et 3%. Cette catégorie de graphe ne permet pas d'émettre davantage de conclusion sur l'origine des erreurs d'apprentissage. En effet, même si ce graphe peut contenir des chaînes allant jusqu'à 12 modèles, le faible nombre d'entrées facilite l'apprentissage et par conséquent réduit l'impact d'une éventuelle rétro-propagation de l'erreur au sein de cette chaîne. Au vu de ces résultats, nous pouvons affirmer que la procédure par apprentissage minimaliste offre des résultats tout à fait satisfaisant pour des modèles contenus dans un graphe "simple". Il est important de rappeler que les résultats présentés sont obtenus à partir des véritables séquences de la simulation DEVS. Ils sont donc les meilleurs résultats que peuvent fournir chacune des procédures d'apprentissage.

Récapitulatif et comparaison des résultats

Afin d'avoir une vision plus claire de l'évolution des erreurs d'apprentissage en fonction du type de graphe et de la procédure utilisée, nous avons concaténé les statistiques d'erreurs obtenues au sein des deux tableaux : 3.1 pour la procédure totale et 3.2 pour la procédure minimaliste. L'apprentissage de la dynamique de chaque modèle atomique qui compose le graphe de modèle offre des très bons résultats. En effet, quel que soit le graphe, l'erreur maximale observée ne dépasse pas les 11% et l'erreur moyenne est d'environ 2.5%. De plus, la médiane inférieure à 2% nous informe que 50% des poids générés admettent moins de 2% ce qui n'est pas négligeable. Même si cette procédure semble être idéale pour garantir un apprentissage de qualité, elle n'est pas envisageable à cause du temps qu'une telle démarche nécessiterait. Observons maintenant les résultats obtenus par application de la procédure minimaliste.

Type de graphes	Min	Max	Médiane	Moyenne
grid-graph	0.0011	10.19	1.851	2.322
tree-graph	0.0008	9.392	1.85	2.251
linked-graph	0.0054	10.31	1.59	2.143
DEVStone_HIM-graph	0.0026	6.469	1.817	2.194

TABLEAU 3.1 – Récapitulatif des statistiques d'erreurs d'apprentissage pour chaque catégorie de graphes par apprentissage total des dynamiques

Le premier constat que nous pouvons faire à partir du tableau 3.2 est que les graphes de même type ("simple" ou "complexe") ont tendance à offrir des résultats similaires. En effet, si l'on compare les résultats du grid-graph et du DEVStone_HIM-graph, ainsi que du tree-graph et du linked-graph, on peut se rendre compte qu'ils sont globalement proches. De plus, en observant d'un peu plus près ces résultats nous pouvons constater que l'apprentissage des modèles atomiques formant les graphes "complexes" par cette procédure est plus difficile que pour les graphes "simples". Ceci prouve que cette approche est sensible à la structure du graphe. En effet, l'erreur moyenne d'apprentissage des graphes "complexes" est presque deux fois plus importante que celle des graphes "simples". De plus, les erreurs maximales enregistrées sont très élevées, même si elle ne représente qu'une infime partie des modèles ($< 10\%$). Ces importantes erreurs trouvent leurs origines dans la taille des chaînes qui composent le graphe, ainsi que sur le nombre d'entrées des modèles.

Type de graphes	Min	Max	Médiane	Moyenne
grid-graph	0.0196	12.68	3.087	3.502
tree-graph	0.01598	41.16	5.511	7.354
linked-graph	0.009	51.09	3.51	8.847
DEVStone_HIM-graph	0.007	11.37	3.096	3.438

TABLEAU 3.2 – Récapitulatif des statistiques d'erreurs d'apprentissage pour chaque catégorie de graphes par apprentissage minimaliste des dynamiques

Ces résultats représentent, en quelque sorte, les meilleurs résultats que peut fournir la procédure minimaliste. Dans la réalité, nous ne disposons pas des séquences obtenues par simulation du modèle DEVS (hormis celle d'un modèle sans entrée). Il est nécessaire de générer ces séquences

pour chaque type de modèle et par conséquent de fournir les entrées dont les modèles ont besoin pour fonctionner. Pour cela, nous utilisons les générateurs d'entrées présentés en section 3.3.1. La section suivante présente d'une part les erreurs d'apprentissage obtenues sur les précédents graphes par application des différents générateurs d'entrées et d'autre part une phase de tests sur des graphes dont la taille est plus proche de la réalité des modèles.

3.4.3 Résultat et estimation du temps d'apprentissage minimaliste

La création d'automates à états probabilistes repose sur l'apprentissage d'une séquence d'observations généralement obtenue par simulation d'un modèle atomique individuel. Comme nous l'avons vu précédemment, les modèles atomiques fournis par le formalisme DEVS peuvent être soumis à des entrées. Celles-ci sont nécessaires pour garantir la création d'une séquence d'apprentissage par simulation. Or, dans le cadre de la simulation, ces entrées sont générées par les sorties des modèles atomiques auxquels ils sont rattachés. Dans notre politique d'apprentissage individuel, nous n'avons pas connaissance de ces sorties. C'est pourquoi il est nécessaire d'utiliser des générateurs d'entrées pour les simuler.

Dans un premier temps, nous allons présenter l'évolution des erreurs d'apprentissage obtenues en fonction des différents générateurs d'entrées, présentés en section 3.3.1, sur les graphes abstraits précédemment utilisés. Rappelons que ces générateurs portent les noms de **GPA** pour générateur purement aléatoire, **GSA** pour générateur par tirage aléatoire parmi une séquence sans entrée et **GSSE** pour générateur d'une séquence sans entrée. Le but étant de déterminer le "meilleur" générateur, au sens de la minimisation des erreurs d'apprentissage. Dans un second temps, nous étendrons cette phase de tests à des graphes de grande taille, pour quelques catégories de graphes abstraits.

Étude des erreurs d'apprentissage pour un grid-graph et pour chaque générateur d'entrées

Cette sous section reprend chacun des graphes utilisés en section 3.4.2 (à l'exception du *DEVS-tone_HIM* qui offre des résultats très proches du *grid-graph*) et présente une étude des erreurs d'apprentissage obtenues en utilisant des générateurs d'entrées. Les résultats sont présentés sous forme de tableaux, à l'image du 3.1, où la première ligne récapitule les statistiques obtenues par apprentissage à partir d'une séquence générée par simulation du véritable modèle DEVS. Le tableau 3.3, présente l'évolution des erreurs d'apprentissage pour chaque générateur d'entrées sur un *grid-graph*.

Type d'apprentissage	Min	Max	Médiane	Moyenne
DEVS	0.0196	12.68	3.087	3.502
GPA	0.7709	26.89	19.82	19.09
GSA	0.5766	19.9	13.73	13.27
GSSE	0.06047	12.72	4.901	5.014

TABLEAU 3.3 – Statistiques d'erreurs d'apprentissage pour un *grid-graph* obtenues par apprentissage minimaliste des dynamiques à partir des différents générateurs d'entrées

Rappelons que cette catégorie de graphes est dite "simple" et nous avons observé en section 3.4.2

que dans ce cas les erreurs d'apprentissage sont les plus faibles. L'utilisation de générateur *GPA* et *GSA* est à l'origine d'une augmentation importante de l'erreur moyenne. En effet, le GPA offre une erreur moyenne de 19% et le GSA de 13% contre seulement 3.5% pour une séquence DEVS. Cette augmentation importante est intimement liée à l'aspect aléatoire du générateur. En effet, même si le GSA génère des valeurs plausibles elles ne respectent pas forcément l'ordre établi par le modèle DEVS. De plus, le générateur GSSE tend à confirmer cette hypothèse. En effet, l'utilisation de celui-ci fournit une erreur moyenne de 5%, ce qui est proche du meilleur résultat espéré.

Étude des erreurs d'apprentissage pour un tree-graph et pour chaque générateur d'entrées

Le tableau 3.4 présente l'évolution des erreurs d'apprentissage pour chaque générateur d'entrées sur un *tree-graph*. Rappelons que cette catégorie de graphe est dite "complexe" et que dans ce cas les erreurs maximums observées sont beaucoup plus importantes.

Type d'apprentissage	Min	Max	Médiane	Moyenne
DEVS	0.01598	41.16	5.511	7.354
GPA	0.04783	54.89	18.65	17.56
GSA	0.02053	42.31	13.41	12.81
GSSE	0.01446	67.45	5.579	10.30

TABLEAU 3.4 – Statistiques d'erreurs d'apprentissage pour un *tree-graph* obtenues par apprentissage minimaliste des dynamiques à partir des différents générateurs d'entrées

Le constat est similaire à celui du *grid-graph*, il y a une augmentation importante de l'erreur moyenne pour les générateurs GPA et GSA. Cependant, dans ce cas l'écart entre l'erreur moyenne obtenue par application du GSA et du GSSE est faible (moins de 3%). En observant d'un peu plus près l'erreur médiane, on peut constater qu'avec le générateur GSSE, la moitié des erreurs sont inférieures à 6% contre 13% pour le GSA. Même si l'erreur maximale est nettement plus importante pour le GSSE (67%) que pour le GSA (42%), les résultats fournis par ce dernier restent moins intéressants que ceux fournis par le GSSE. Rappelons que ce type de graphe fournit de grandes chaînes de modèles atomiques. Or dans ce cas nous ne constatons pas d'explosion des erreurs d'apprentissage, même par utilisation de générateurs aléatoires. Il semblerait donc que la taille des chaînes ait moins d'impact que le nombre d'entrées des modèles atomiques sur l'évolution de l'erreur. Confirmons ceci par l'étude des erreurs d'apprentissage d'un *linked-graph*.

Étude des erreurs d'apprentissage pour un linked-graph et pour chaque générateur d'entrées

Le tableau 3.5 présente l'évolution des erreurs d'apprentissage en fonction du type de générateur d'entrées pour un *linked-graph*. Rappelons que ce type de graphe possède des modèles atomiques pouvant comporter jusqu'à 26 entrées, et des chaînes moyennement longues (15 dans ce cas). Commençons par rappeler que cette catégorie de graphes est celle qui offre les plus grosses erreurs d'apprentissage dans un cas idéal (avec séquence par simulation du modèle DEVS) comme le montre la première ligne du tableau ci-dessus. Contrairement aux autres types de graphes, les erreurs d'apprentissage obtenues par utilisation d'un générateur GSA sont plus importantes que

celles obtenues pour un GPA (erreur moyenne de 36% contre 25%). En observant les erreurs maximales, on peut constater que les apprentissages des modèles atomiques de cette catégorie de graphes semblent très mal supporter l'aspect aléatoire des générateurs d'entrées. En effet, les générateurs GPA et GSA offrent respectivement des erreurs maximales de 784% et de 1524% qui correspondent aux modèles à 26 entrées. La présence de modèles possédant un très grand nombre d'entrées fait exploser les erreurs d'apprentissage lors de l'utilisation de générateur d'entrées reposant sur un processus aléatoire. Cette hypothèse se justifie par l'observation de l'erreur maximale obtenue par application d'un générateur GSSE, qui n'est que de 51%. Dans ce cas, on peut constater que les erreurs sont très proches de celles obtenues à partir d'une séquence DEVS.

Type d'apprentissage	Min	Max	Médiane	Moyenne
DEVS	0.009	51.09	3.51	8.847
GPA	0.0068	784.4	12.23	25.62
GSA	0.0241	1524	9.19	36.5
GSSE	0.02178	51.25	3.64	8.916

TABLEAU 3.5 – Statistiques d'erreurs d'apprentissage pour un *linked-graph* obtenues par apprentissage minimaliste des dynamiques à partir des différents générateurs d'entrées

Au vu de l'ensemble des résultats présentés, il semble tout à fait judicieux d'utiliser un générateur de type GSSE pour simuler les entrées des modèles atomiques. L'utilisation de celui-ci n'impacte que très légèrement la qualité de l'apprentissage, contrairement aux deux autres types de générateurs. Ces résultats tendent à justifier l'hypothèse émise en section 3.3.1 concernant la dépendance des modèles atomiques par rapport à une séquence générée par un modèle sans entrée. L'observation des résultats de cette sous section montre l'importance de l'ordre d'apparition des événements sur la dynamique d'un modèle. En effet, même si le générateur GSA respecte les générations d'événements en terme de fréquence, il ne les respecte pas en terme de chronologie. Or, c'est cette même chronologie qui influe directement la dynamique des modèles. En prenant en compte cette remarque, nous avons tenté de mettre en place une autre politique d'apprentissage. Celle-ci consiste à ne plus apprendre un HMM par nombre d'entrées, mais par combinaisons d'entrées. C'est à dire qu'on différencie un modèle possédant une entrée de type $S0$ (séquence d'entrée provenant d'un modèle sans entrée) d'un modèle possédant une entrée de type $S1$ (séquence d'entrée provenant d'un modèle à une entrée). La figure 3.23 illustre ce typage de modèles par un exemple simple.

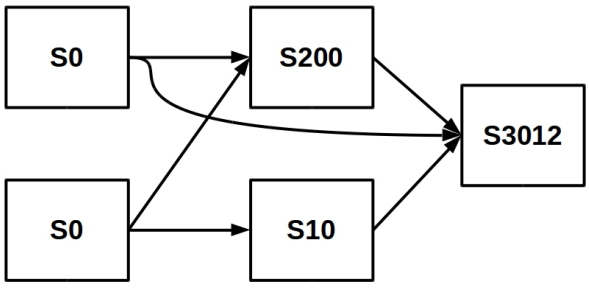


FIGURE 3.23 – Exemple de typage des modèles atomiques d'un graphe en fonction de leurs entrées

Chaque modèle atomique du graphe de modèles se voit attribuer un type de la forme $Sabcd\cdots$, où S correspond au nom de la dynamique, a est le nombre d'entrées du modèle et b, c, d, $\cdots$ sont les nombres d'entrées des modèles en amont. Par exemple, sur le graphe de la figure 3.23, le modèle atomique de droite porte le type $S3012$ car c'est un modèle de dynamique S à 3 entrées, où les modèles auxquels il est rattaché ont respectivement 0, 1 et 2 entrées.

Cette approche à l'avantage d'offrir un apprentissage plus spécifique pour chaque modèle. Cependant, le nombre d'apprentissage total nécessaire pour garantir l'exploration de tous les cas possibles se voit considérablement augmenter. Dans le cas des graphes "complexes", celui-ci tend vers le nombre de modèles atomiques qu'il contient. Or, au vu des résultats offerts pour le générateur GSSE, il serait grotesque de perdre une grande quantité de temps en apprentissage pour gagner moins de 10% d'erreurs. C'est la raison pour laquelle cette démarche fut abandonnée.

Conclusion

La pondération du graphe de modèles est indispensable au bon déroulement de son partitionnement. En effet, l'information contenue par celle-ci doit refléter au mieux les dynamiques qui unissent les modèles atomiques. Cette pondération nécessite l'apprentissage de chaque dynamique qui compose le modèle DEVS que l'on souhaite distribuer. Pour garantir un partitionnement de qualité, il est nécessaire de fournir d'une part une méthode de partitionnement de qualité et d'autre part une pondération elle aussi de qualité. Cette qualité est totalement dépendante du choix de la méthode d'apprentissage et du type d'automate que l'on souhaite apprendre. Afin de faire un choix judicieux, il est important de mettre en perspective les informations disponibles pour réaliser les apprentissages. Dans notre cas, nous cherchons à estimer la durée moyenne entre deux émissions de message. Il est donc nécessaire de disposer de l'ensemble des durées entre chaque émission de message des différents modèles. Cette information est facilement récupérable par le biais de la fonction d'avancement du temps t_a du formalisme DEVS. La base d'observations à partir de laquelle est appris chaque dynamique est obtenue par simulation d'un modèle atomique individuel. Le formalisme DEVS autorise les modèles atomiques à posséder des entrées. Celles-ci peuvent perturber la dynamique interne d'un modèle, à tel point que deux modèles (un sans entrée, l'autre avec entrées) ne vont plus générer les même sorties. Il est donc important de réaliser un apprentissage par nombre d'entrées pour chaque dynamique qui compose le graphe de modèle DEVS (procédure minimaliste).

Au vu du type de données disponibles et de l'utilisation faite de l'automate appris, nous avons opté pour l'utilisation de chaînes de Markov cachées. En effet, ce type de structure offre de très bons algorithmes d'apprentissage et des algorithmes de génération de séquences statistiquement proches de celles apprises. L'intérêt de l'apprentissage repose sur le fait de créer un graphe d'automate à états probabilistes reflétant au mieux les séquences d'émission de messages. En effet, la simulation d'un tel graphe ne nécessite aucun calcul, contrairement à la simulation. Ce qui permet d'obtenir une durée moyenne fiable sans être contraint de simuler l'intégralité du graphe de modèles. Pour garantir une pondération de qualité, il est très important d'apprendre au mieux chaque dynamique. Pour cela, nous avons mis au point un processus allant de l'apprentissage à la génération de séquence d'observations en utilisant les HMM. Ce processus est résumé par le schéma

bilan de la figure 3.24.

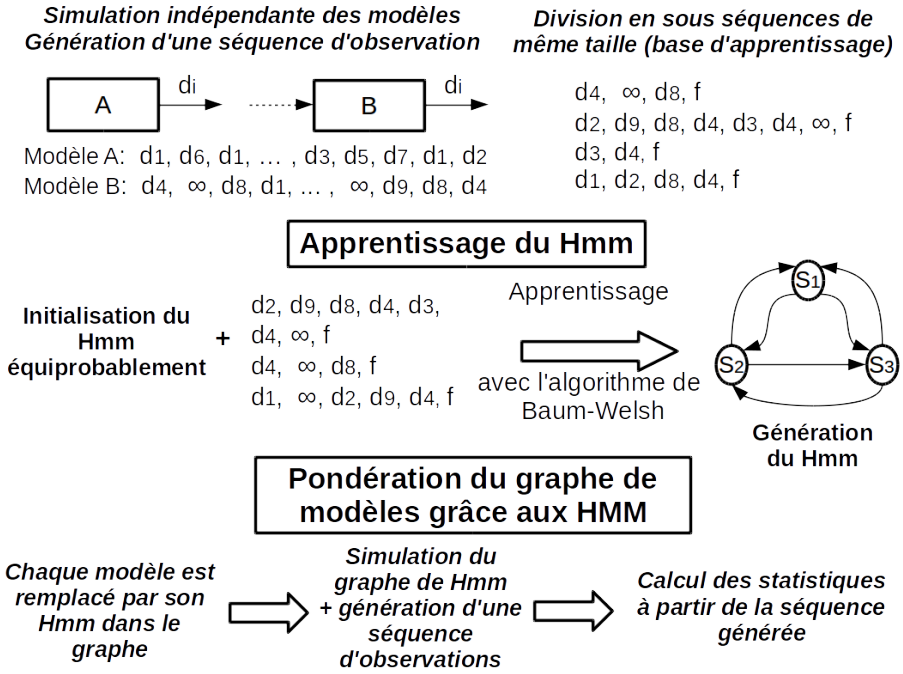


FIGURE 3.24 – Schéma bilan du processus d'apprentissage : de la génération de séquence d'apprentissage, à la pondération du graphe de modèles

La nature des modèles atomiques liée au formalisme DEVS a nécessité la mise en place de certaines adaptations, notamment en ce qui concerne les algorithmes de générations de séquences. La prise en compte de ces particularités est indispensable à la création de HMM permettant la génération de séquences statistiquement proches de celles obtenues par simulation. L'une d'entre elles est notamment la nécessité de créer des générateurs d'entrées, indispensable pour la génération de séquence d'apprentissage des modèles à entrées multiples. Nous avons pu constater au cours de la section 3.4.2 que seul le générateur GSSE offre des résultats d'apprentissage satisfaisants. C'est pourquoi, nous avons opté pour l'utilisation de ce type de générateurs pour nos futures phases de tests. De plus, l'étude des erreurs d'apprentissage de modèles appartenant à différentes catégories de graphes (pour une même dynamique de base) nous a permis de montrer l'impact qu'a le nombre d'entrées sur la qualité de l'apprentissage par procédure minimaliste. En effet, plus un modèle à un nombre d'entrées est important, plus il est difficile à apprendre. Ceci est sans compter sur la taille des chaînes, qui elles aussi ont un impact sur la qualité de la séquence générée par les algorithmes de génération de séquences.

Chapitre 4

Validation de la démarche : tests et résultats

L'optimisation des temps de simulation parallèle et distribuée reposent sur un certain nombre de facteurs étroitement liés. En effet, d'un point de vue purement théorique, il est possible de visualiser cette optimisation comme étant un problème de partitionnement contraint, à l'image de ce qui a été présenté en section 1.3. Or, l'optimalité de la simulation repose en majorité sur l'efficacité du noyau de simulation. En effet, la transcription de la qualité du partitionnement ne peut être effective qu'à condition que le noyau de simulation soit idéalement conçu. Le rôle de cette section est, en premier lieu, de présenter le noyau de simulation utilisé pour réaliser nos simulations, mais aussi de présenter les différentes architectures matérielles sur lesquelles le noyau a été testé.

La restructuration de la hiérarchie de modèle, à l'aide du partitionnement de graphe, et la mise en place d'un noyau parallèle ont permis de réaliser des tests de performances dans un contexte parallèle. Ces tests, présentés en section 4.2, ont pour objectif de montrer l'impact que peut avoir l'architecture matérielle, ainsi que la qualité du partitionnement sur les temps de simulation pour des modèles partiellement asynchrones. En effet, il est bon de rappeler que la qualité du partitionnement prédétermine l'efficacité d'une parallélisation ou d'une distribution, notamment pour les modèles synchrones. C'est pourquoi, cette section présente une étude comparative des résultats de partitionnement en fonction du choix de la méthode et du type de pondération sur un panel de graphes présentés en section 3.4.1.

Pour terminer, la section 4.4 présente l'évolution des speedups dans un contexte distribué. Contrairement à la parallélisation, le transfert d'événements entre les nœuds de calcul a un coût important lié à l'utilisation du réseau. Cette sous-section a pour objectif premier de mettre en évidence toute l'importance de minimiser ce coût de transfert, pour ne pas impacter le gain lié à l'équilibre de charge. Et dans un second temps, de montrer que l'efficacité du partitionnement n'est pas le seul gage d'une distribution de qualité. En effet, lorsque les modèles sont totalement asynchrones, il est très difficile, voire impossible, de créer une partition de modèles où les *IMM* sont équilibrés à chaque pas de temps.

4.1 Présentation des outils liés à la simulation

La qualité d'une simulation n'est pas exclusivement liée à son temps d'exécution, elle est aussi liée au fait de garantir une exactitude des résultats (phase de simulation) et une correspondance avec le phénomène réel simulé (phase de modélisation). La cohérence et l'efficacité d'une simulation résident en partie dans son noyau de simulation. En effet, une simulation ne peut être valide si le noyau à partir duquel cette dernière est réalisée ne l'ai pas. Cette section présente notre noyau expérimental de simulation inspiré de nos expériences avec le projet VLE [QDR09], ainsi que l'ensemble des outils et du matériel utilisés pour réaliser nos expérimentations.

4.1.1 Présentation de la structure du noyau de simulation Parallel-DEVS

La simulation repose sur l'utilisation d'un noyau de simulation Parallel-DEVS écrit en C++ 11, qui est présenté en section 4.1.1. Celui-ci est un noyau expérimental pour l'étude des évolutions potentielles du projet VLE (Virtual Laboratory Environement) [QDR09], qui est un environnement de multimodélisation et de simulation reposant lui aussi sur le formalisme DEVS.

Le noyau de simulation Parallel-DEVS se compose en trois parties ayant chacune des rôles bien distincts. Le fonctionnement, ainsi que les différentes classes nécessaires au fonctionnement du noyau de simulation et à sa chaîne de fabrication, sont présentés par la figure 4.1. La première partie, nommée ***paradevs***, permet la création et la gestion des différentes classes abstraites (voir 1.2.3) nécessaires au déroulement d'une simulation Parallel-DEVS. La seconde partie gère tout ce qui concerne la création du graphe et son partitionnement, via la librairie Boost graph présentée en section 4.1.3 et à partir des informations fournies par paradevs. La dernière partie gère tout ce qui touche à la pondération du graphe, à l'aide des outils d'apprentissage présentés en section 3 et de la dynamique des modèles contenue dans paradevs.

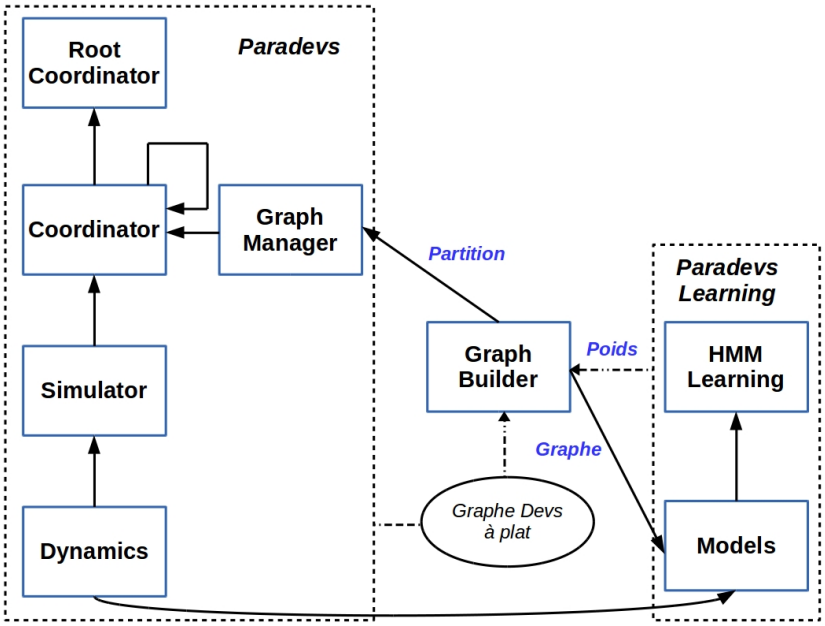


FIGURE 4.1 – Schéma du noyau de simulation Parallel-DEVS

Description du noyau de simulation paradevs

La simulation Parallel-DEVS est régie au minimum à l'aide des quatre classes suivantes :

- ***Dynamics***, comme son nom le suggère, permet de définir l'ensemble des fonctions qui animent les modèles atomiques.

- ***Simulator*** met en oeuvre la logique DEVS au niveau des modèles atomiques. On associe à chaque simulateur une dynamique.

- ***Coordinator*** est la classe permettant de gérer les coordinateurs. C'est cette dernière qui gère les liens de dépendance entre les simulateurs appartenant à un même coordinateur via la classe GraphManager. Lors de la restructuration de la hiérarchie, c'est elle qui reçoit les informations concernant le partitionnement contenues par la classe GraphManager. Elle gère également la construction et l'évolution des échéanciers.

- ***RootCoordinateur*** gère l'évolution de la simulation par un échange de dates d'activation contenues par la classe Coordinator.

La classe GraphManager permet de faire le lien entre le partitionnement obtenu par le biais de la classe GraphBuilder et le noyau de simulation. En effet, lors de la restructuration de la hiérarchie, le graphe de modèles se voit découpé en k sous-graphes. Chacun de ces sous-graphes se voit distribué sur un nœud de calcul. Le rôle du GraphManager est de gérer les connexions de cette nouvelle hiérarchie. Pour cela, chaque partie de la partition doit fournir les connexions internes à la parties (*graphs*), l'ensemble des connexions sortantes (*output_edges*), l'ensemble des connexions entrantes (*input_edges*) et pour finir l'ensemble des connexions sortantes permettant de faire le lien entre les coordinateurs (*parent_connections*). La figure 4.2 illustre ces notions.

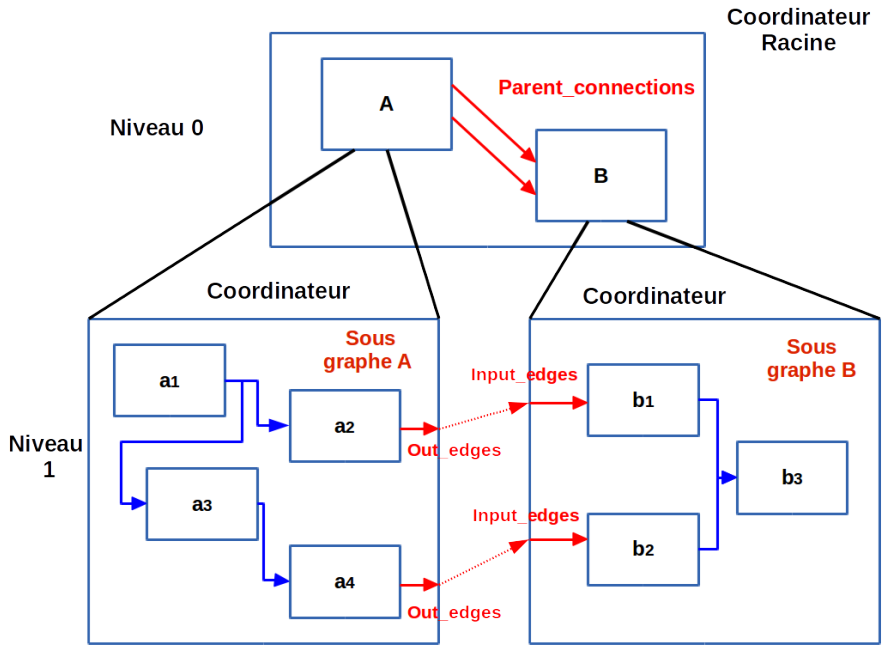


FIGURE 4.2 – Illustration du rôle de GraphManager

Dans l'exemple de la figure 4.2, la nouvelle hiérarchie se compose de deux coordinateurs (A et B) possédant respectivement quatre et trois modèles atomiques. Le rôle du GraphManager est de faire le lien entre les simulateurs d'une même partie, entre les simulateurs de parties différentes, mais aussi entre les coordinateurs. Dans cet exemple, les différents paramètres fournis par le GraphManager sont fournis par le tableau 4.1.

Modèle A			
graphs	***output_edges***	***input_edges***	***parent_connections***
(a_1 , a_2) (a_1 , a_3) (a_3 , a_4)	(a_2 , b_1) (a_4 , b_2)		$((0 , a_2) , (1 , b_1))$ $((0 , a_4) , (1 , b_2))$
Modèle B			
graphs	***output_edges***	***input_edges***	***parent_connections***
(b_1 , b_3) (b_2 , b_3)		(a_2 , b_1) (a_4 , b_2)	

TABLEAU 4.1 – Exemple d'informations contenues par la classe GraphManager pour un partitionnement en deux parties

Le noyau de simulation distribué

Le noyau de simulation Parallel-DEVS en version distribué, illustré par la figure 4.3, repose sur la même hiérarchie de classes que la version non distribuée. Seuls les coordinateurs sont spécialisés et la notion de LogicalProcessor fait son apparition dans l'architecture. En effet, pour gérer la communication entre le coordinateur racine et les coordinateurs en charge des différentes parties, le protocole MPI (Message Passing Interface) est mis en œuvre. Du coté du coordinateur racine et donc au niveau du noeud le supportant, chaque sous-coordinateur est représenté et est lié à un proxy. Ce proxy a pour rôle de préparer et d'envoyer les événements via MPI. La transformation des événements utilise un mécanisme de "sérialisation" des objets C++ disponible au travers de la librairie boost::serialization. Sur chaque nœud de calcul prenant en charge un sous-modèle distribué, un objet spécifique, le "LogicalProcessor", fait le travail inverse. Il attend les messages réseaux contenant les événements DEVS, le "désérialisent" et les communiquent aux coordinateurs. Lorsque les modèles atomiques gèrent des événements externes, s'ils ont pour destinataire un ou plusieurs modèles appartenant à un autre modèle couplé alors les événements font le chemin dans l'autre sens.

Les coordinateurs du nœud principal (contenant le coordinateur racine) sont tous multithreadés afin de permettre un traitement parallèle des événements dans chacun des nœuds de calcul. Pour les expériences purement parallèles, l'architecture est très semblable : seuls les objets proxy et LogicalProcessor sont absents. Naturellement, les coordinateurs des sous-modèles ne sont pas dupliqués. Les techniques pour le multithreading sont multiples : un pool de threads ou un système par envoi asynchrone de message, ... en ne citant qu'eux. Pour notre part, nous avons fait le choix de l'envoi asynchrone de messages. Pour cela, nous nous sommes inspirés du code développé dans [Wil12].

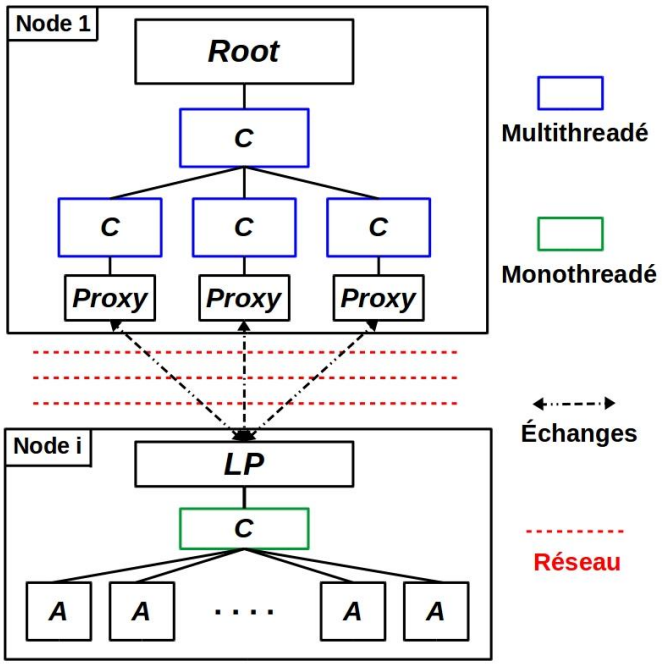


FIGURE 4.3 – Exemple de schéma d'un noyau de simulation distribuée en trois parties, où une seule partie est représentée par soucis de clarté.

4.1.2 Présentation de l'architecture matérielle

Afin de réaliser nos tests et simulations, nous disposons d'un ensemble d'architectures matérielles qui se compose de :

— Processeur Intel Core i5-2520M - 2.50GHz : 2 cœurs avec 4 threads (en mode hyper-threadé)

— Processeur Intel(R) Core(TM) i7-3840QM - 2.8 GHz : 4 cœurs avec 8 threads (en mode hyper-threadé)

— Processeur Samsung Exynos 5422 avec un Cortex A15 2.0 Ghz quad core et un Cortex A7 quad core : 4 gros coeurs et 4 petits coeurs

— Réseau de 4 processeur Intel Xeon CPU E5-4603 v2 - 2.20GHz à mémoire partagée

Ces architectures offrent différentes possibilités en terme de cœurs et de nombre de threads allouables. La plus petite architecture possède une configuration à deux cœurs offrant le plus petit facteur d'accélération à quatre threads (2C + 2H), lié au processeur Intel Core i5-2520M. La seconde configuration double la capacité (4C + 4H), en offrant 4 coeurs et la possibilité d'allouer jusqu'à huit threads. La troisième configuration offre une configuration hybride (4C + 4LC) avec 4 gros cœurs et 4 autres cœurs moins puissants, pour un total de huit threads allouables. Celle-ci est légèrement plus puissante que la précédente. Et la dernière configuration permet d'explorer une architecture multi-processeur / multi-cœur, offrant un total de 32 coeurs et la possibilité d'allouer jusqu'à 64 threads.

Chacune de ces architectures a été utilisée dans le cadre des simulations parallèles présentées en section 4.2. En ce qui concerne les simulations distribuées présentées en section 4.4, elles ont été réalisées à partir de l'avant-dernière architecture matérielle représentée par la figure 4.4.

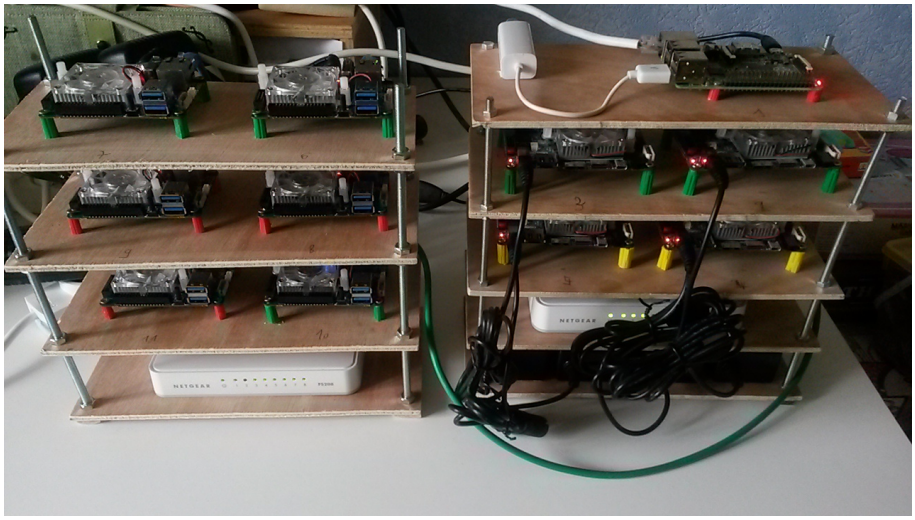


FIGURE 4.4 – Photo du cluster utilisé pour réaliser les simulations distribuées

4.1.3 Outils de programmation liés au graphe

La création ainsi que la manipulation des graphes se fait à l'aide de la librairie ***Boost Graph*** (BGL). Celle-ci permet de manipuler les éléments d'un graphe, accessibles via différentes classes telles que :

- ***vertex*** qui permet de définir et de manipuler les sommets du graphe
- ***edge*** qui permet de définir et de manipuler les arcs du graphe
- ***graph*** qui permet de déclarer le graphe en lui même

Ces classes sont les briques fondamentales permettant de manipuler un graphe via ***Boost Graph***. À l'image des informations présentées en section 1.3.1 et 1.4.3, BGL permet de visualiser le graphe de différentes façons :

- à l'aide d'une liste d'adjacence par le biais de la classe ***adjacency_list***
- à l'aide d'une matrice d'adjacence par le biais de la classe ***adjacency_matrix***

BGL offre également la possibilité de manipuler le graphe, non pas par rapport aux sommets, mais plutôt grâce aux arcs du graphe. Cette visualisation du graphe est possible par le biais de la classe ***edge_list***. Dans notre cas, nous avons opté pour la liste d'adjacence qui est construite à partir des paramètres suivants :

adjacency_list < OutEdgeList, VertexList, Directed, VertexProperties, EdgeProperties, GraphProperties, EdgeList >

où, *OutEdgeList* est une structure contenant la liste des arcs sortant de chaque sommet, *VertexList* et *EdgeList* sont respectivement les listes des sommets et des arcs qui composent le graphe auquel est attaché la liste d'adjacence. BGL offre également la possibilité de choisir l'orientation du graphe via son paramètre *Directed* comme suit :

- ***undirectedS***, le graphe est non orienté
- ***directedS***, le graphe est orienté

- ***bidirectionalS***, le graphe est orienté par des arcs à double sens

Les paramètres *VertexProperties*, *EdgeProperties* et *GraphProperties* sont des structures permettant de fournir des informations complémentaires aux sommets et arcs du graphe, ainsi qu'au graphe lui même. En effet, dans notre cas, chaque sommet doit posséder un index et un poids, et chaque arc possède un poids. Ainsi, il est possible d'accéder à ces informations directement à partir de l'objet en question. En plus de fournir toutes ces structures complètes, BGL fournit également un large éventail de fonctions et d'algorithmes permettant de manipuler efficacement les graphes. Cette librairie est donc un atout considérable, tant sur le plan de l'implémentation que de l'efficacité. C'est pourquoi nous avons fait le choix de travailler à partir de celle-ci.

Les calculs matriciels nécessaires au déroulement de la méthode spectrale ont été réalisés à partir de la librairie ***Eigen***. Celle-ci est une bibliothèque d'analyse numérique codée en C++ et développée par Benoît Jacob et Gaël Guennebaud à l'INRIA.

4.2 Première architecture : en mode parallèle

Dans le cadre de la réduction des temps de simulation, un certain nombre de modifications du noyau de simulation ont été réalisées afin de pouvoir aboutir à la simulation parallèle et distribuée. En effet, une étude de la structure de celui-ci a permis, dans un premier temps, de réduire le temps d'ordonnancement des événements en optant pour un simple découpage de l'échéancier. Cette idée fut la base de la restructuration de la hiérarchie de modèles, présentée en section 2.1.1. La mise en place d'un noyau de simulation parallèle voit apparaître une nouvelle problématique liée à la parallélisation des bags, ainsi qu'à leur taille. Le rôle de cette section est de présenter cette problématique, qui s'avère être parfaitement valide pour la simulation distribuée, et de réaliser une étude permettant de valider le comportement du noyau parallèle. Face à des modèles DEVS de plus en plus nombreux et coûteux en temps de calcul, l'utilisation d'une seule machine en mode parallèle peut ne plus suffire à garantir un temps de simulation convenable. En effet, le nombre de threads allouables sur une seule machine peut, d'une part, ne pas suffire à contenir l'intégralité des modèles atomiques (faute de mémoire suffisante) et, d'autre part, ne pas suffire à garantir une distribution optimale des modèles atomiques d'une simulation de grande taille. L'objectif de cette démarche est de trouver une piste de solution au problème de parallélisation des bags, dans le but de la généraliser dans le cadre de la simulation distribuée.

4.2.1 Premiers pas : le découpage des échéanciers

Comme nous l'avons vu en section 1.2.5, une simulation DEVS peut posséder une structure dite "à plat", où chaque modèle atomique est attaché à un unique modèle couplé faisant le lien avec le coordinateur racine. Or, pour gérer la synchronisation entre les simulateurs (modèles atomiques) et les coordinateurs (modèles couplés), chaque coordinateur doit posséder son propre échéancier. Celui-ci contient les dates de réveil des modèles atomiques attachés à ce coordinateur. Ces dates sont calculées à l'aide de la fonction d'avancement du temps t_a. Nous comprendrons aisément toute l'importance des échéanciers pour le déroulement d'une simulation. En effet, c'est à partir de ces échéanciers que la simulation évolue. Le choix d'une structure "à plat" d'un modèle DEVS a

pour conséquence d'avoir une simulation qui ne repose que sur l'évolution d'un unique échéancier de grande taille.

Avant même d'entamer une procédure de parallélisation des modèles atomiques, nous nous sommes interrogés sur l'impact qu'ont la taille et le type de structure utilisée pour créer ces échéanciers sur le temps de simulation. En effet, au cours d'une simulation, un échéancier est sollicité un très grand nombre de fois, notamment pour fournir la prochaine date de réveil et mettre à jour la prochaine date de réveil. Lorsqu'un événement externe vient perturber la dynamique interne d'un modèle atomique, l'échéancier se voit généralement modifié et subit des opérations telles que des tris et des insertions. De part la nature des modèles atomiques, ce genre de situation est généralement très fréquent. Nous nous sommes donc interrogés, d'une part sur le type de structure à utiliser pour accueillir les échéanciers et d'autre part s'il ne serait pas plus judicieux de manipuler n sous échéanciers de petite taille plutôt qu'un seul de grande taille.

Structure d'accueil des échéanciers

À l'origine, notre choix de structure pour accueillir nos échéanciers s'est porté sur un *vector* de la librairie *std* du langage C++. En effet, cette structure simple d'utilisation, offre des fonctionnalités répondant parfaitement à nos attentes (tri, insertion, retrait, ...). Cependant, après plusieurs séries de tests sur des exemples de grande taille, nous nous sommes aperçus que l'utilisation de cette structure n'est pas judicieuse pour représenter nos échéanciers. En effet, pour un nombre de modèles atomiques supérieur à 10000, le temps de simulation par utilisation des échéanciers de type *vector* "explosent" (si on fait abstraction du temps de calcul de chaque modèle). Cette explosion provient probablement du coût lié aux différentes fonctions que fournit la structure *vector*. Celui-ci permet un accès aux données en temps constant ($O(1)$), une insertion en $O(1)$ si l'élément est ajouté en queue et en $O(N)$ s'il est ajouté en tête et un tri en $N \log N$. Pour palier ce problème, nous avons cherché une autre structure offrant de meilleures complexités de tri et d'insertion. Notre choix s'est porté sur une structure du type *Boost*::*Heap*, qui offre de meilleures complexités. En effet, là où un *vector* offre une fonction de tri à complexité $N \log N$, le *Heap* offre une complexité inférieure ou égale à $\log N$, sachant que cette dernière n'est obtenue que dans les pires cas. Ce gain est lié à la structuration interne des données et aux algorithmes de calcul reposants sur l'algorithme de Fibonnaci. Cependant, rappelons que la plupart des fonctions fournies par ce type de structure sont dépendantes de la taille de l'objet à manipuler. Même si les échéanciers de type *Heap* sont bien plus performants que ceux de type *vector*, ils restent toujours dépendants du nombre de modèles atomiques que contient la simulation.

Influence de la subdivision des échéanciers sur le temps de simulation

Notre première approche d'optimisation des temps de simulation, avant de parler de parallélisation ou de distribution, consistait à réaliser un découpage optimal du graphe de modèles dans le but de remplacer l'unique échéancier de grande taille par k sous échéanciers de taille réduite. Cette idée est la base de notre recherche sur l'optimisation des simulations distribuées présentée en section 2.1.1. En effet, la nouvelle hiérarchie de modèles était initialement conçue pour garantir

la création efficace des sous-échéanciers. La figure 4.5 illustre le découpage de l'échéancier.

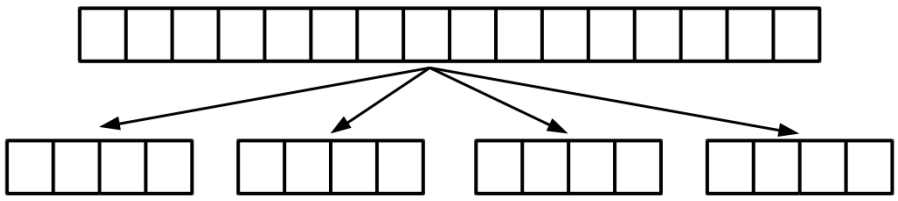


FIGURE 4.5 – Exemple de subdivision d'un échéancier

Nos premiers travaux ont permis de montrer que le simple fait de subdiviser l'échéancier permet de gagner du temps d'ordonnancement. On fait donc abstraction des temps de calcul liés au modèle de simulation. En effet, dans ce cas, le choix des structures est primordial. La figure 4.6 présente les résultats obtenus pour un découpage d'échéancier entre 1 et 100, pour des tree-graph de tailles comprises entre 1000 et 16000.

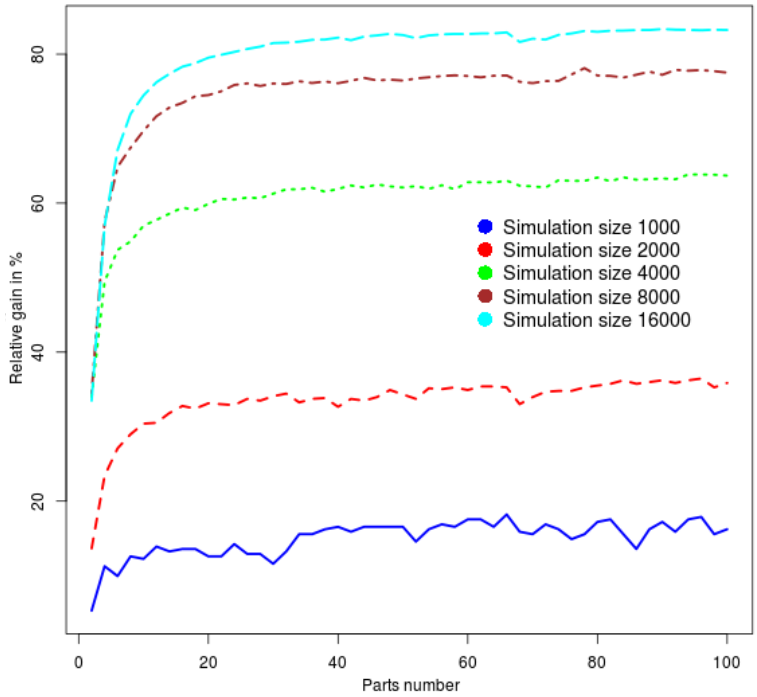


FIGURE 4.6 – Évolution du gain relatif en fonction du nombre de sous-échéanciers et du nombre de modèles atomiques

Cette figure montre que le simple fait de manipuler plusieurs échéanciers de petite taille, plutôt qu'un unique de grande taille, permet de réduire considérablement les temps d'ordonnancement des événements. Cette tendance semble être amplifiée par le nombre de modèles atomiques qui composent la simulation. Plus ce nombre est grand, plus le gain obtenu tend vers 80%. Cette observation peut avoir un impact fondamental sur les temps de simulation où les modèles atomiques possèdent un faible de temps de calcul. Dans un contexte parallèle ou distribué, les simulations se composant de modèles atomiques à faible temps de calcul peuvent atteindre des performances bien plus élevées grâce à ce simple découpage. Ces remarques ont été les prémices d'une recherche

orientée sur la simulation distribuée. Cependant, notre première approche fut de s'intéresser à la parallélisation.

4.2.2 Qu'est ce que le speedup d'une simulation ?

La simulation parallèle à l'image de la simulation distribuée, nécessite un maximum de modèles atomiques exécutés simultanément pour garantir une simulation optimale en temps, mais elle n'est que peu impactée par la transmission de messages entre les modèles de par leur proximité physique (des coeurs au sein d'un même processeur ou des processeurs sur une même carte avec des bus rapides). En effet, le coût réseau lié à la transmission de messages est quasi nul en mode parallèle. Il existe une notion permettant de quantifier le facteur d'accélération apporté par une parallélisation efficace des modèles atomiques : elle se nomme speedup (du mot accélération en anglais).

Pour une transition donnée, le speedup se calcule en effectuant le rapport entre la somme des modèles atomiques actifs et le nombre maximum de modèles actifs au sein d'un des coordinateurs. Ce qui se traduit mathématiquement par :

$$\text{speedup}_j = \frac{\sum_{i=1}^{k} n_i}{\max_{i \in \{1,\ldots,k\}} n_i}$$

où j est l'indice de la transition, k est le nombre de coordinateurs et n_i le nombre de modèles actifs dans le coordinateur i.

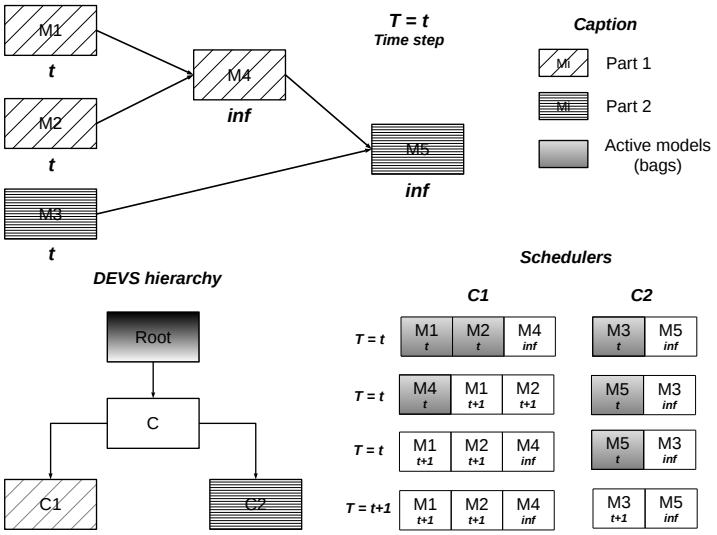


FIGURE 4.7 – Exemple d'exécution d'une simulation à cinq modèles atomiques et évolution du speedup à chaque transition

Les modèles actifs sont les modèles appartenant aux ensembles IMM, présentés et définis en section 1.2.4, lors du traitement de la fonction de transition. Pour une date donnée, la taille des IMM varie selon la dynamique des modèles atomiques. Si cette taille est la même alors le speedup

théorique est égal au nombre de coordinateurs actifs, car il y a un thread par coordinateur et qu'en fonction de l'architecture matérielle, tous les threads peuvent être exécutés en parallèle modulo l'accès à la mémoire. Cette notion de speedup est illustrée par un exemple dans la figure 4.7.

Ce schéma présente différentes informations d'une simulation au temps t. Le schéma du haut est une représentation du graphe de modèles partitionné en deux parties. Sa représentation hiérarchique est donnée en bas à gauche. À chaque coordinateur est associé un échéancier. Leur évolution pour chaque transition est illustrée en bas à droite. Les modèles actifs sont représentés par les cases grisées. Les cases grisées d'un même coordinateur forment un IMM. Pour la première transition (M_1, M_2) et (M_3) forment deux IMM, le speedup associé à cette transition est donc de $\frac{2+1}{2} = \frac{3}{2} = 1.5$. Pour que les performances soient optimales, il est nécessaire que le nombre de modèles par IMM soit maximal à chaque transition. Dans ce cas, le speedup tend vers le speedup théorique absolu. En effet, plus la taille des IMM est importante, plus il y aura de modèles atomiques exécutés simultanément à chaque pas de temps. Le speedup peut être calculé à chaque transition. Dans l'exemple de la figure 4.7, il y a 3 transitions de speedup respectifs 1.5, 2 et 1 et donc un speedup de simulation à 1.5. Ce qui signifie qu'à chaque transition, il y a en moyenne 1.5 modèles atomiques exécutés, contre 2 espérés dans le meilleur des cas. La condition sous-jacente est que les temps de calcul de chaque modèle sont équivalents. On y reviendra dans le cadre de l'équilibrage de charge.

Pour un pas de temps donné, n_t speedups sont calculés (un par transition). Il est donc possible de déduire le speedup correspondant à une date t en calculant la moyenne du speedup à chaque transition :

$$\text{speedup}(t) = \sum_{j=1}^{n_t} \text{speedup}_j$$

En conservant les speedups obtenus à chaque pas de temps, nous sommes en mesure de déterminer le speedup global de la simulation par :

$$\text{speedup} = \sum_{t=1}^{t_{max}} \text{speedup}(t)$$

où t_{max} est le temps total de la simulation.

Ce speedup est naturellement très lié à la hiérarchie des coordinateurs/simulateurs. C'est pourquoi il est important de créer des sous-modèles équilibrés lors de la restructuration de la hiérarchie. Cependant, ce seul équilibre n'est pas suffisant pour garantir une parallélisation optimale. En effet, il est nécessaire d'avoir un équilibre entre les IMM à chaque transition pour garantir un équilibre parfait. Dans notre cas, tous les modèles ont la même charge en terme de calcul, c'est pour ça qu'on parle de nombre de modèles et non pas de poids. Pour être efficace, une simulation parallèle se doit d'avoir un speedup proche du nombre de threads disponibles. Par la suite, ce speedup portera le nom de *speedup théorique absolu*. Nous verrons qu'en fonction de la nature de la dynamique des modèles atomiques et du type d'architecture utilisé, il n'est pas toujours possible d'atteindre ce speedup.

4.2.3 L'impact de la taille des bags sur le speedup

Cette sous-section présente une étude de l'évolution du speedup d'une simulation parallèle pour les graphes de types *grid* et *tree*, présentés en section 3.4.1, à partir des architectures matérielles présentées en section 4.1.2. Les résultats de simulation présentés dans cette sous-section sont obtenus pour un temps de calcul d'environ 2 ms par modèle atomique sur l'architecture à base d'Odroid. Contrairement à une simulation parfaitement synchrone où chaque modèle effectue une tâche à chaque pas de temps, la simulation DEVS fournit à chaque modèle atomique sa propre dynamique (liée à la dynamique interne et au nombre d'entrées). Il est donc rare de travailler à partir d'une simulation où chaque modèle est parfaitement synchrone. De ce fait, l'obtention d'un speedup proche du speedup théorique absolu semble être parfaitement utopique, dans un contexte de simulation parallèle ou distribuée en mode pessimiste. C'est pourquoi, nous avons fait le choix de comparer le speedup obtenu au speedup théorique correspondant au meilleur résultat que l'on peut espérer en prenant en compte la dynamique des modèles atomiques.

Influence de l'architecture matérielle sur le speedup

Les figures 4.8 et 4.9 comparent l'évolution du speedup pour chacune des architectures matérielles, présentées en section 4.1.2, au speedup théorique. Pour chacun de ces graphes, on étudie l'évolution du speedup en fonction du nombre de threads, que nous faisons varier de deux à douze. Ainsi cette étude permet d'une part de voir l'importance de la dynamique des modèles atomiques sur l'évolution du speedup et d'autre part l'importance qu'a l'architecture matérielle sur ce dernier.

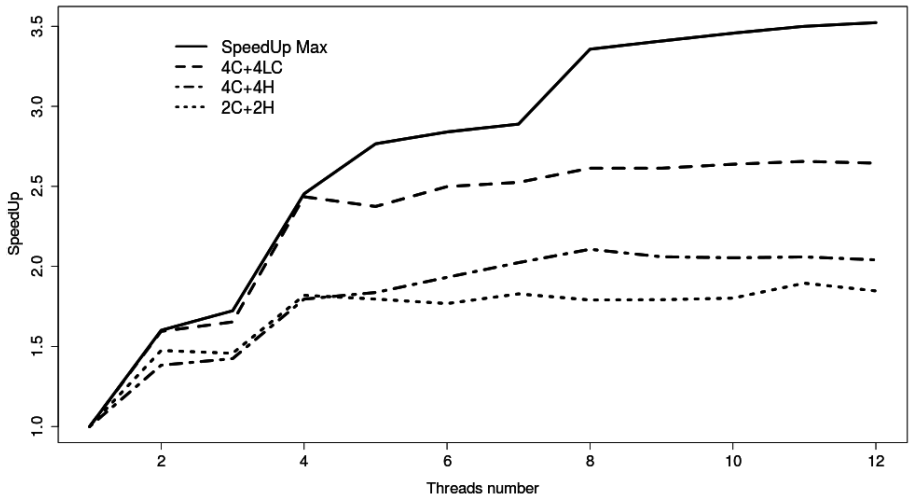


FIGURE 4.8 – Évolution du speedup pour un tree-graph à partir de trois architectures matérielles

Ces courbes montrent l'influence de l'architecture matérielle sur le speedup mais aussi l'efficacité de l'implémentation C++. En effet, tant que le nombre de threads est inférieur ou égal au nombre de coeurs, le speedup est proche, voire très proche, de la valeur théorique, comme on peut l'observer sur la figure 4.9. Pour l'architecture 4C+4LC, on observe une légère inflexion à partir de 8 threads car les 4 coeurs supplémentaires sont moins performants que les 4 premiers coeurs.

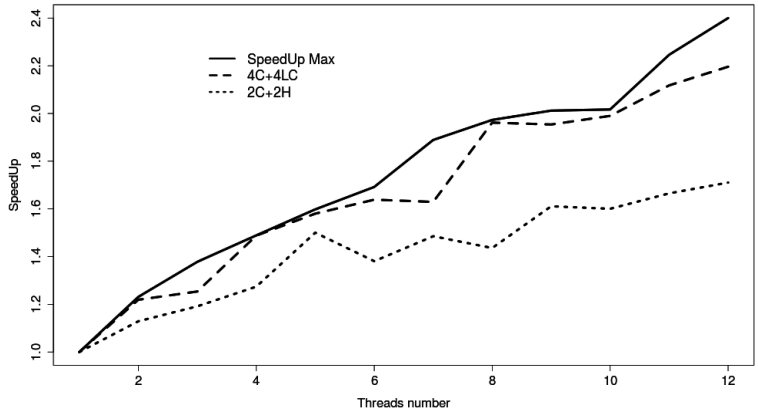


FIGURE 4.9 – Évolution du speedup pour un grid-graph à partir de deux architectures matérielles

Les résultats présentés ont été réalisés à partir d'un graphe non pondéré, avant de mettre en place la procédure de pondération par apprentissage. Ce qui pourrait expliquer la difficulté rencontrée par certaines architectures moins puissantes.

Influence de la qualité du partitionnement sur l'évolution du speedup et inversement

Les résultats présentés dans cette sous-section sont tous obtenus à partir de graphe de taille 1000 et pour une structure hiérarchique à 4 sous modèles. La figure 4.10 présente l'évolution du speedup à chaque transition au pas de temps $t = 0$ sur un tree-graph. On peut observer que la fonction de transition est calculée 63 fois et que durant une petite partie de ce laps de temps un seul modèle est actif (*IMM* de taille 1). Ce qui se traduit en réalité par une simulation non parallèle durant cette période (à condition que cette situation soit la même à chaque pas de temps). Près de la moitié des transitions offrent un speedup satisfaisant (speedup proche de 4) et une vingtaine d'entre elles offrent un speedup compris entre 1.5 et 2.5. Étant donnée la structure du graphe, ce résultat semble satisfaisant.

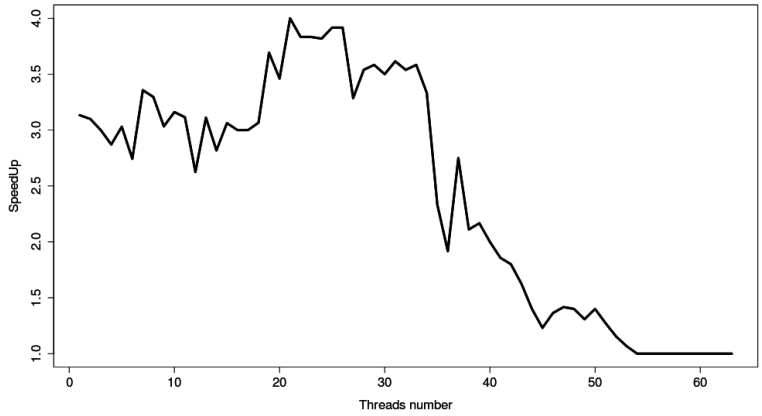


FIGURE 4.10 – Évolution du speedup à $t = 0$ pour chaque transition sur un tree-graph de taille 1000

Dans le cas d'un grid-graph, présenté par la figure 4.11, la conclusion semble différente. Tout d'abord, on est très loin d'obtenir le speedup théorique absolu de 4 (2.2 ici), correspondant aux quatre threads. Cela s'explique par la dynamique des modèles du grid-graph. En effet, dans cet

exemple, les événements se propagent par vague du coin supérieur gauche au coin inférieur droit. Or, le partitionnement étant partiellement influencé par une dynamique spatiale, le graphe se retrouve découpé en 4 sous-grilles presque régulières. Ce qui semble en contradiction avec la dynamique de la simulation. Le nombre de sous-modèles simultanément calculés peut être au maximum de 3 (dans de rares cas) et à cause de ce partitionnement, le nombre de modèles actifs se retrouve à 1 pour la moitié des transitions. Cependant, la structure particulière de ce graphe fait que le speedup ne peut pas toujours être égal à celui du speedup théorique absolu car le nombre de modèles parallélisables est, par moment, plus faible que le nombre de threads disponibles. C'est notamment le cas en début et en fin de simulation.

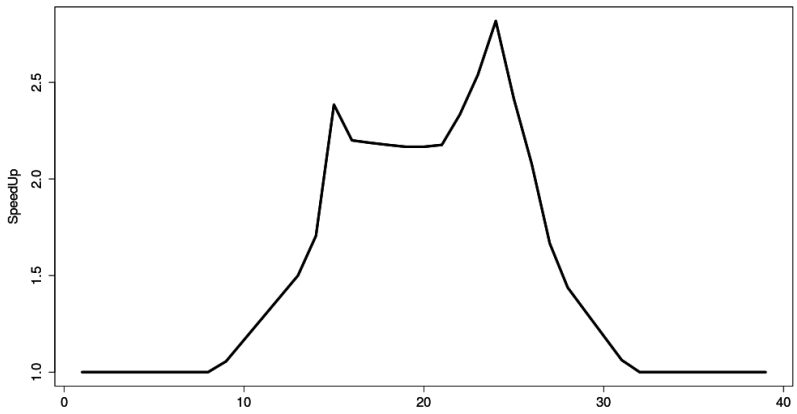


FIGURE 4.11 – Évolution du speedup à $t = 0$ pour chaque transition sur un grid-graph de taille 1000

En observant la figure 4.12, correspondant à l'évolution du speedup théorique sur ce même graphe, on peut constater que le partitionnement semble bel et bien être à l'origine de ce speedup de faible qualité. En effet, dans ce cas, le speedup théorique admet à de nombreuses reprises 4 modèles actifs simultanément et que dans de très rares cas un unique modèle actif. Globalement, en observant cette figure, on peut espérer un speedup proche de 3 pour un partitionnement adéquat et en supposant que l'information soit disponible au préalable. En effet, il est bon de rappeler que le partitionnement est réalisé en amont de la simulation. Il n'y a donc aucune information disponible sur une éventuelle taille d'*IMM*.

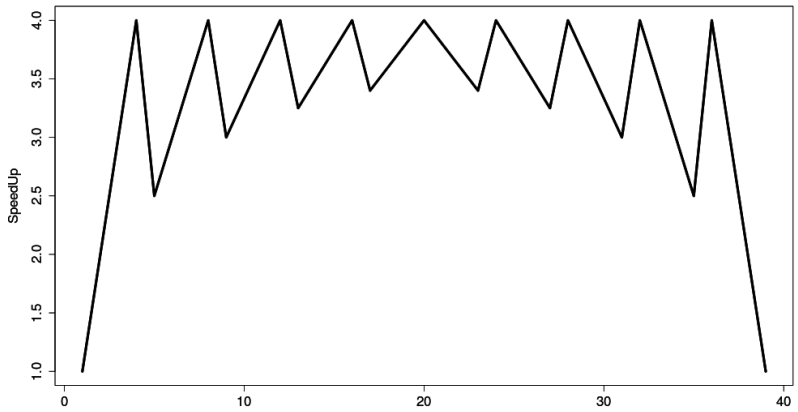


FIGURE 4.12 – Évolution du speedup théorique à $t = 0$ pour chaque transition sur un grid-graph

Pour se convaincre du phénomène, il suffit de comparer un partitionnement obtenu par application de la méthode GGGP à un partitionnement aléatoire. En effet, le partitionnement par GGGP est régi en partie par un aspect spatial, contrairement à la méthode aléatoire. On parle ici d'aléatoire au sens de la répartition des modèles atomiques. Cependant, chaque partie contient un même nombre de modèles et par conséquent chaque thread fournit la même charge de calcul. La figure 4.13 montre l'évolution du speedup en fonction du nombre de threads alloués et du type de méthode de partitionnement utilisé. On constate que, dans ce cas, quelque soit le nombre de threads alloués la simulation restructurée à partir d'une méthode GGGP offre toujours un meilleur speedup qu'une méthode aléatoire. Même si le speedup n'atteint pas le nombre de threads disponibles, en raison des spécificités liées à DEVS et de la dynamique des modèles, ce résultat montre l'intérêt d'un partitionnement réfléchi.

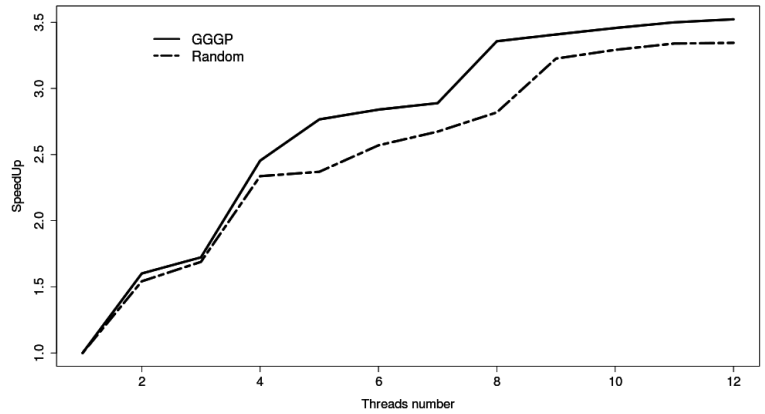


FIGURE 4.13 – Évolution du speedup d'une simulation restructurée à partir de deux méthodes de partitionnement : GGGP et aléatoire

Dans l'absolu, la parallélisation idéale consiste à obtenir un speedup égal au nombre de threads disponibles durant la quasi totalité de la simulation. La partition générée doit permettre aux résultats de tendre vers un tel speedup pour être de qualité. Or, cet objectif est généralement difficile à atteindre à cause des spécificités liées à DEVS. En effet, certaines simulations sont dites asynchrones, c'est à dire que les modèles atomiques qui composent la simulation susceptibles de posséder des t_a différents pour chaque modèle et/ou prenant des valeurs différentes selon l'état du modèle. Or, cette particularité rend très difficile l'obtention d'une partition garantissant la création de *IMM* de même taille, où le nombre de modèles actifs au sein des *IMM* doit correspondre au nombre de threads disponibles. Il est important de rappeler que lors du partitionnement, nous ne connaissons pas la "nature" des modèles atomiques, seul l'équilibrage de charge et la minimisation des coûts de transfert sont pris en compte. La prise en compte de la dynamique des modèles atomiques reste cependant indispensable à la bonne répartition des modèles au sein des nœuds de calcul dans un contexte distribué, où la quantité d'information en transit via le réseau est importante.

4.3 Tests de performance des méthodes de partitionnement

Cette section présente un ensemble de tests dont l'obectif est d'évaluer le gain apporté par la mise en place des différentes optimisations. Dans un premier temps, nous allons étudier l'impact de la mise en place d'une pondération du graphe de modèles sur la qualité du partitionnement. Cette sous-section fait référence aux procédures de pondération présentées en section 3.1. Dans un second temps, nous allons étudier le gain apporté par la mise en place des optimisations de la méthode GGGP, présentées en section 2.2. Et pour terminer, nous allons présenter une étude comparative des résultats de partitionnement obtenus par application des méthodes GGGP et spectrale. L'objectif de cette section est de mettre en avant la plus-value obtenue par application de nos optimisations.

4.3.1 L'impact de la qualité de la pondération du graphe de modèles sur le partitionnement

L'information contenue dans la pondération du graphe a une importance fondamentale pour garantir l'obtention d'une partition de qualité. En effet, une pondération erronée, ou absente, peut être à l'origine d'une divergence de l'algorithme de partitionnement et par conséquent, empêcher la minimisation du coût de coupe. Même si dans ce cas, l'équilibre de charge reste correct, cette divergence peut être à l'origine d'une perte de temps considérable liée à un échange trop important de données entre les modèles de différents nœuds de calcul. Cependant, cette remarque est uniquement valable si l'impact des échanges de messages n'est pas négligeable par rapport au temps d'exécution des modèles. Le rôle de cette sous-section est de présenter une étude de l'impact de la pondération sur le partitionnement, mais aussi et surtout de montrer l'impact de la qualité de celle-ci sur le partitionnement. En effet, une pondération de faible qualité peut être à l'origine de la création d'une partition moins bonne que celle d'un graphe sans pondération.

Type graphe	Nombre de parties	2	4	8	16	32
Grid	**SPSM**	2.04117	4.02845	8.14262	12.7703	20.6683
	APSM	2.03868	4.02191	8.09735	12.7575	20.5809
	DPSM	2.0941	4.0069	8.15722	12.953	20.2015
Grid	**SPAM**	2.04117	4.13719	8.24921	12.8588	20.7199
	APAM	2.03501	4.49074	8.72177	13.5491	20.1954
	DPAM	2.0891	4.59627	8.64169	12.8871	20.8287
Tree	**SPSM**	0.738793	3.03226	5.05016	10.6614	17.8086
	APSM	1.14726	2.95293	4.88809	10.7636	17.2343
	DPSM	1.14726	2.92898	4.79003	10.0657	16.4026
Tree	**SPAM**	1.53948	3.53536	5.38533	13.6635	19.0882
	APAM	1.22962	3.05932	4.50324	11.4737	17.1478
	DPAM	1.22962	3.16958	5.38047	10.2105	17.7559

TABLEAU 4.2 – Comparaison des résultats de partitionnement avec et sans pondération pour des grid-graph et tree-graph de taille 300

Les tableaux 4.2 présentent les différents coûts de coupes obtenus sur deux graphes de type

grid et tree de différentes tailles (petite et grande), suivant différentes politiques de pondération. En effet, cette étude porte d'une part sur l'influence de la politique de pondération du graphe sur la qualité du partitionnement. Les différentes politiques de pondération utilisées sont les suivantes : aucune pondération (**SP**), pondération à partir des poids obtenus par apprentissage (**AP**) et pondération par les poids réels obtenus par simulation DEVS (**DP**). D'autre part, cette étude a pour objectif de montrer l'impact de la pondération sur l'application d'une méthode multi-niveaux, présentée en section 1.5. En effet, il est important de rappeler que la première phase de cette méthode consiste à contracter le graphe de façon à minimiser le nombre d'arcs de poids fort. Or l'absence de pondération modifie la contraction du graphe et impacte par conséquent le partitionnement. Dans certains cas, ce manque d'information pourrait avoir tendance à dégrader la qualité du coût de coupe et dans d'autre cas, il pourrait ouvrir l'accès à des solutions inexploitables par le biais du graphe pondéré et des contraintes liées à la méthode de partitionnement. Afin de mettre en avant ces phénomènes, nous allons partitionner un graphe de taille 300, d'une part sans appliquer une méthode Multi-niveaux (**SM**) et d'autre part en lui appliquant cette dernière pour un niveau de contraction de 200 (**AM**). L'ensemble de ces informations sont concaténées dans le tableau 4.2 par le biais d'une nomenclature **abcd**, où les deux premiers caractères informent du type de pondération utilisée parmi celles présentées précédemment, et les deux derniers informent sur l'utilisation ou non d'une méthode Multi-niveaux.

Chaque cas est étudié dans le but d'évaluer l'impact que peut avoir la pondération sur la qualité du partitionnement en fonction de la méthode utilisée mais aussi en fonction de la nature du graphe. C'est pourquoi nous travaillons à partir d'un graphe "simple" (grid-graph) et d'un graphe "complexe" (tree-graph). Le tableau 4.2 présente uniquement les différents coûts de coupe obtenus par application d'une méthode GGGP optimisée pour chacun des plans d'expériences précédemment présentés. Cependant, celui-ci ne présente pas l'évolution des balances de partitionnement car la pondération des arcs du graphe n'affecte pas l'évolution de celle-ci au cours du partitionnement.

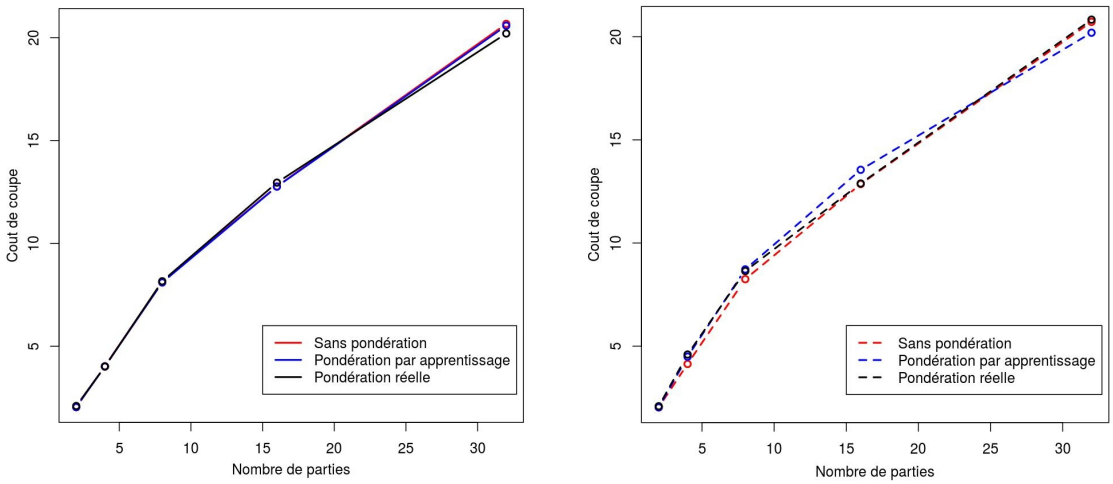


FIGURE 4.14 – Comparaison des coûts de coupe en fonction des types de pondération du grid-graph. Sans multi-niveaux à gauche, avec multi-niveaux à droite.

Afin de pouvoir comparer la qualité du partitionnement d'un graphe non pondéré à celui d'un graphe pondéré, nous avons projeté la partition du graphe non pondéré sur ce dernier dans le but d'en déduire le véritable coût de coupe. En partant du principe qu'un graphe sans pondération (SP) est fortement susceptible de fournir un partitionnement de mauvaise qualité et que la pondération apprise (AP) à partir d'un générateur d'entrée de type GSSE sera forcément de moins bonne qualité que la véritable pondération obtenue par simulation (DP), on peut s'attendre à obtenir une certaine logique dans les résultats. En effet, en théorie, les coûts de coupe obtenus par application d'une même méthode de partitionnement sont sensés respecter cet ordre : DP < AP < SP, le meilleur coût de coupe étant celui obtenu à partir du graphe pondéré à partir des données de simulation. Le tableau 4.2 nous montre que dans la pratique ce n'est pas toujours le cas. Détaillons un peu plus l'étude de ces résultats. La figure 4.14 présente graphiquement l'évolution des coûts de coupe, en fonction de la pondération, pour le grid-graph. Sans utilisation du Multi-niveaux, nous pouvons constater que les résultats des trois pondérations sont proches, même si ceux du DPSM ne sont pas toujours les meilleurs. En effet, la structure régulière du graphe (en forme de grille) influence beaucoup la nature du partitionnement, ce qui rend presque inutile la pondération dans ce cas. Rappelons tout de même que la méthode GGGP repose sur une notion de voisinage et qu'elle est donc influencée, en partie, par la forme du graphe. Dans le cadre de l'application de la méthode Multi-niveaux sur ce graphe, nous pouvons nous rendre compte que dans la majeure partie des cas, le partitionnement du graphe non pondéré offre de meilleurs résultats. Il semblerait que ce cas soit un exemple où le manque de pondération influence la contraction qui influence elle-même l'algorithme de partitionnement GGGP et lui donne accès à des solutions que ne peut pas trouver ce dernier lorsque le graphe est pondéré. Au vu de ces résultats, il semblerait peu judicieux de réaliser un apprentissage des graphes à géométrie simple, car la pondération apportée par celui-ci n'apporte aucun gain lors du partitionnement.

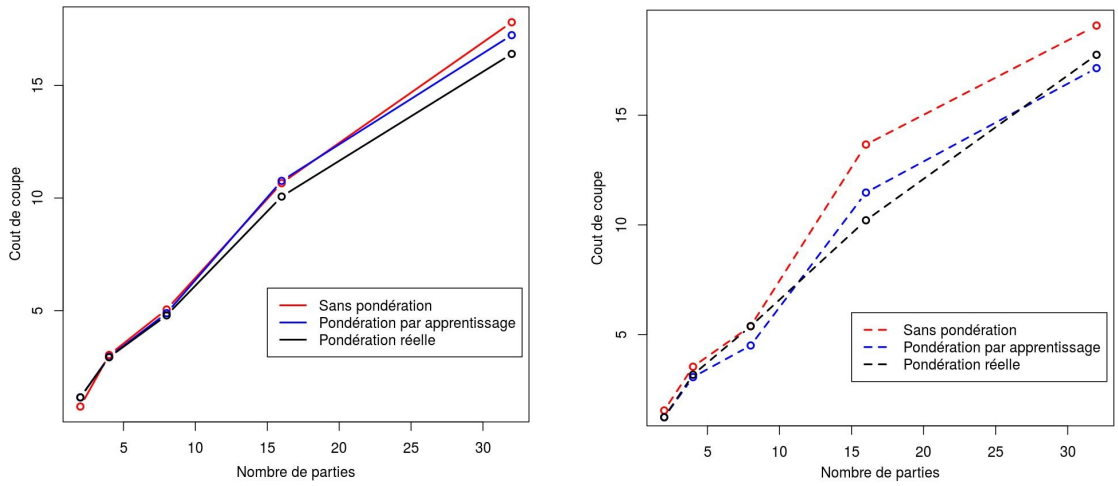


FIGURE 4.15 – Comparaison des coûts de coupe en fonction des types de pondération du tree-graph. Sans multi-niveaux à gauche, avec multi-niveaux à droite.

Intéressons nous maintenant aux résultats de partitionnement obtenus sur un graphe à géométrie complexe : un tree-graph. Ces résultats sont fournis par les deux dernières cases du tableau

4.2, où la première présente les résultats obtenus sans appliquer une méthode Multi-niveaux et la seconde présente les résultats obtenus en appliquant cette dernière. Ils sont également illustrés par la figure 4.15. Dans ce cas, les résultats sont totalement différents. En effet, en observant les coûts de coupe obtenus sans Multi-niveaux, on peut observer que dans la majeure partie des cas le partitionnement du graphe non pondéré, les résultats sont moins bons que celle des deux autres pondérations. De plus, l'ordre logique de classement des coûts de coupe auxquels nous nous attendions semble être avéré dans ce cas. Si on observe l'évolution des coûts de coupe en fonction du nombre de parties, nous pouvons nous rendre compte que l'écart entre les approches s'amplifie. Ce qui semble de bonne augure pour nos distributions. Les résultats obtenus par application du Multi-niveaux, à quelques contre-exemples près, montrent l'importance qu'a la pondération sur la contraction et sur le partitionnement. En effet, on peut observer que l'écart des coûts de coupe est plus important que lorsque le Multi-niveaux n'est pas utilisé. Cela montre bien que la pondération a un rôle très important sur la contraction.

4.3.2 Évaluation du gain apporté par les optimisations de la méthode GGGP

Le rôle de cette section est de montrer l'impact des améliorations de la méthode GGGP sur la qualité du partitionnement. Rappelons que cette qualité dépend de deux valeurs : d'une part de la balance de partitionnement qui doit être inférieure à 1.05 (et de préférence tendre vers 1) et d'autre part du coût de coupe qui doit être le plus faible possible. La méthode GGGP a pour objectif de déterminer la bissection équilibrée de coût de coupe minimum. Or, dans notre amélioration, nous avons autorisé la relâche de la balance à 1.05, comme indiqué par la littérature. Cette relâche permet, dans certains cas, d'obtenir un meilleur coût de coupe au prix d'une très légère dégradation de l'équilibre de la partition. Rappelons également, que la méthode GGGP initialement conçue ne "force" pas le respect de la balance lorsqu'un graphe n'est pas connexe (voir section 2.2.1). Les tableaux 4.3 et 4.4 présentent l'évolution conjointe du coût de coupe et de la balance de partitionnement pour deux graphes de types grid et tree, pour un nombre de parties variant de 2 à 16 et par application de la méthode GGGP optimisée et non optimisée.

Type GGGP	**Grid 300**			
	2	**4**	**8**	**16**
Opti	2.0916	4.0855	8.2547	13.019
balance	1.0034	1.0103	1.0242	1.0519
Non Opti	[2.0971 - 4.6048]	[5.1065 - 8.4028]	[9.0935 - 12.511]	[14.827 - 18.672]
balance	1.0034	1.01038	1.0242 - 1.7716	1.0519 - 1.4948
Type GGGP	**Tree 300**			
	2	**4**	**8**	**16**
Opti	1.0911	2.7949	5.5859	10.958
balance	1.0066	1.0266	1.0133	1.0133
Non Opti	[1.1594 - 2.2542]	[2.2406 - 5.9485]	[4.4586 - 8.6411]	[9.6782 - 13.566]
balance	1	1 - 1.9733	1.2 - 3.6	1.44 - 3.36

TABLEAU 4.3 – Comparaison des résultats de partitionnement par application d'un méthode GGGP optimisée et non optimisée sur deux catégories de graphe de taille 300

Le tableau 4.3 présente les résultats de partitionnement obtenus sur deux graphes de taille 300 sans utiliser de méthode Multi-niveaux. Les résultats contenus dans la première ligne de ce tableau sont obtenus par application de notre méthode GGGP améliorée et ceux contenus dans la seconde sont obtenus par application d'une méthode GGGP standard. Nous pouvons constater que les résultats obtenus dans la première ligne sont uniques. En effet, pour dix applications d'une méthode GGGP améliorée sur un même graphe, une seule partition est obtenue. Au contraire, la seconde ligne offre un éventail de solutions car la méthode GGGP telle qu'elle a été conçue tire au hasard un unique sommet de départ dans l'ensemble des sommets du graphe. Or ce sommet a un impact très important sur la qualité de la partition. Pour obtenir ces résultats, nous avons initialisé un bon nombre d'applications de la méthode GGGP par différents sommets de départ. Les résultats obtenus sont présentés sous forme d'intervalles contenant le plus petit et le plus grand coût de coupe observé. De même, pour les balances de partitionnement. Pour le partitionnement du grid-graph, la méthode GGGP optimisée offre toujours un meilleur résultat que ceux obtenus à partir des n applications de la méthode GGGP standard. De plus, contrairement à cette dernière, notre approche garantit toujours un équilibre de la partition (ou tout du moins un résultat très proche de notre critère). En effet, à partir d'un 8-partitionnement, on peut constater que la méthode GGGP standard rencontre des difficultés pour garantir le respect de l'équilibre de la partition.

Intéressons nous maintenant au tree-graph. De part sa structure "complexe", ce dernier est nettement plus sensible à l'application itérative d'une méthode de partitionnement. Cette affirmation semble se confirmer par les résultats fournis par la méthode GGGP standard. En effet, à partir d'un 4-partitionnement, la méthode GGGP standard n'est plus en moyen de garantir l'obtention d'une partition équilibrée (balance > 1.15). Un tree-graph possède une structure particulière favorisant l'apparition de graphes non connexes au cours des applications itératives de la méthode GGGP (à l'image de la figure 2.9 de la section 2.2.3). Certes, de cette façon, la méthode garantit l'obtention de coût de coupe plus faibles que la méthode améliorée, mais la partition s'en voit lourdement déséquilibrée. À l'image de la 8-partition qui peut offrir un coût de coupe d'environ 4.5 mais avec une balance minimale de 1.2. L'application de notre optimisation de la méthode GGGP sans faire appel à une méthode multi-niveaux, permet d'obtenir une partition de coût de coupe faible tout en respectant notre contrainte principale d'équilibrage de charge.

Le tableau 4.4 présente les résultats de partitionnement obtenus sur deux graphes de taille 10000. L'intérêt de ces exemples reposent sur le fait qu'il est nécessaire d'appliquer une méthode multi-niveaux pour réaliser un partitionnement. En effet, au vu de la taille du graphe et de la contrainte de taille nécessaire à la bonne convergence d'une méthode GGGP (voir 1.4.4), l'utilisation de la méthode multi-niveaux est indispensable. Les résultats obtenus sur le grid-graph sont très satisfaisants. En effet, le coût de coupe obtenu par application de la méthode GGGP améliorée se situe toujours dans la borne inférieure de l'intervalle de solutions fournies par la méthode GGGP standard, tout en garantissant l'obtention d'une balance équilibrée. Il est important de rappeler que dans de rares cas, la méthode GGGP standard peut offrir un meilleur résultat. Cependant, celle-ci est obtenue au prix de nombreuses expérimentations, avec une très faible fréquence d'apparition. De même que pour le graphe 300, la méthode GGGP standard rencontre des difficultés à respecter l'équilibre de la partition dès que le nombre de parties dépasse les 8.

Type GGGP	Grid 10000			
	2	**4**	**8**	**16**
Opti	14.619	32.411	61.452	101.07
balance	1.0062	1.0368	1.0464	1.0512
Non Opti	[15.469 - 23.414]	[30.375 - 42.817]	[58.991 - 70.2861]	[93.219 - 109.26]
balance	1.0212 - 1.05	1.004 - 1.05	1.0352 - 1.327	1.0456 - 1.6832
Type GGGP	**Tree 10000**			
	2	**4**	**8**	**16**
Opti	1.8018	3.6767	7.3321	16.583
balance	1.0386	1.048	1.0448	1.0464
Non Opti	[1.8013 - 2.2601]	[2.8587 - 4.0498]	[4.1751 - 7.8172]	[8.7688 - 13.0683]
balance	1.0002 - 1.0386	1.1548 - 1.2028	1.2104 - 2.2272	1.8784 - 3.4288

TABLEAU 4.4 – Comparaison des résultats de partitionnement par application d'un méthode GGGP optimisée et non optimisée sur deux catégories de graphe de taille 10000

Pour le tree-graph, les résultats sont légèrement différents. En effet, nous pouvons constater qu'au-delà de 4 parties, la méthode GGGP optimisée ne garantit plus l'obtention d'une partition de coût de coupe inférieur à la borne supérieure de l'intervalle obtenu par application de la méthode GGGP standard. Cependant, il est important d'observer également l'évolution de la balance de partitionnement. Dans le cas de la méthode GGGP standard, aucun partitionnement ne fournit de partition équilibrée (au-delà de deux parties). Or, contrairement à cette dernière, la méthode GGGP améliorée conserve un équilibre quelque soit le nombre de parties souhaitées. C'est le respect de la contrainte d'équilibre qui est à l'origine d'une partition de coût de coupe plus grand que celui obtenu par application de la GGGP standard. Cependant, au vu des résultats, il semble impossible d'obtenir un coût de couple plus faible tout en respectant l'équilibre de la partition. Ces résultats montrent toute la force de nos améliorations de la méthode GGGP, qui par une étude importante du spectre de solutions permet de satisfaire au mieux nos deux contraintes.

4.3.3 Étude comparative de performance entre les méthodes GGGP et Spectrale

Les méthodes GGGP et spectrale reposent sur deux mécanismes de partitionnement totalement différents. En effet, l'une utilise la notion de voisinage pour se propager au sein du graphe et l'autre repose sur l'algèbre linéaire et la résolution de systèmes. De plus, de par leur fonctionnement, la méthode GGGP est initialement conçue pour résoudre un problème contraint et la méthode spectrale pour résoudre un problème non contraint. Cependant, l'utilisation de l'une et l'autre est parfaitement possible pour chacun des problèmes de partitionnement au prix de quelques ajustements. Dans le cadre de notre problématique d'optimisation des temps de simulation distribuée, nous souhaitons réaliser un partitionnement de type contraint pour garantir l'obtention d'une partition équilibrée. Il semblerait donc plus avisé d'utiliser une méthode conçue à cet effet. Le but de cette partie est, d'une part, d'évaluer la qualité du partitionnement obtenu par application d'une méthode spectrale dans un cas contraint en comparant les résultats de cette dernière à une application dans un contexte non contraint. Et d'autre part, de comparer les résultats de partitionnement obtenus par application d'une méthode GGGP améliorée et d'une méthode spectrale.

Étude du comportement de la méthode spectrale

La méthode spectrale, présentée en section 1.4.3, peut être utilisée directement lorsque la taille du graphe ne dépasse pas les 7000 sommets. En effet, au-delà d'un tel nombre, nous rencontrons un problème mémoire pour contenir l'intégralité des matrices disponibles. De plus, le temps nécessaire à la résolution du système $M_{Lap}\, x = \lambda x$ est fortement dépendant du nombre de sommets qui composent le graphe. Il est donc beaucoup plus rapide d'appliquer une méthode spectrale sur un graphe de taille 1000 que sur un graphe de taille 7000. Ces informations nous poussent à coupler l'utilisation de la méthode spectrale à celle d'une méthode Multi-niveaux. Au vu des éléments émis en section 1.4.3, nous avons opté pour un appel itératif hiérarchique de la méthode spectrale pour obtenir un k-partitionnement. L'utilisation de la méthode spectrale au sein d'une méthode multi-niveaux nous pousse aussi à modifier le critère de classification des sommets au sein du second vecteur de Fiedler. En effet, pour respecter la problématique de partitionnement contraint, il est nécessaire de garantir l'obtention d'une bissection équilibrée à chaque itération. Or, la proposition émise en section 1.4.3, consistait à trier le second vecteur de Fiedler par ordre croissant et d'attribuer à la première partie la moitié des sommets et à l'autre la seconde moitié. Lors de l'application d'une méthode multi-niveaux, les sommets du graphe sont fusionnés ce qui modifie leur poids. On se retrouve donc en présence d'un graphe contracté où les sommets n'ont pas tous le même poids. Il est donc important de prendre en compte cette information lors de la classification des modèles en deux parties. La classification ne s'effectue pas en fonction du nombre de sommets mais en fonction du poids moyen : tant que le poids moyen n'est pas atteint, on ajoute les sommets à la seconde partie.

La méthode spectrale est initialement conçue pour obtenir une partition non-contrainte (voir 1.3.4). L'adaptation de celle-ci au partitionnement contraint a un coût sur la qualité de la partition. Cette sous-section a pour objectif d'évaluer ce coût et de proposer une démarche afin de le réduire. Pour cela observons les informations contenues dans le tableau 4.5. Ce tableau fournit l'évolution du coût de coupe et de la balance pour le partitionnement d'un tree-graph de taille 10000 en quatre parties, pour différents niveaux de contraction par application d'une méthode spectrale. Celui-ci se compose en deux parties, la première présente les résultats obtenus par application de la méthode spectrale pour résoudre un problème non-contraint (voir 1.3.4) et la seconde pour résoudre le problème contraint.

Partitionnement	**Niveau de contraction**				
	200	**500**	**1000**	**3000**	
Contraint	6.0729	5.2724	8.8045	15.7803	***cut***
	1.0396	1.0268	1.0172	1.0002	***balance***
Non Contraint	1.20564	1.21507	1.21507	1.60939	***cut***
	2.374	2.3712	2.3712	2.3768	***balance***

TABLEAU 4.5 – Évolution de la qualité d'un 4-partitionnement en fonction du niveau de contraction et du problème de partitionnement sur un tree-graph de taille 10000

Les résultats présentés dans ce tableau sont obtenus pour une taille de graphe inférieure à 3000, car au-delà le temps nécessaire pour réaliser le partitionnement devient trop important. De

plus, comme le présente la seconde partie du tableau, dans le cadre du partitionnement contraint, plus le niveau de contraction est faible (ce qui implique un graphe de grande taille) plus le coût de coupe est de faible qualité. Cependant, nous pouvons constater que l'évolution de la balance subit l'effet inverse : plus le niveau de contraction est faible, plus la balance est proche de 1. Au vu de ces résultats, il semblerait judicieux de fixer le niveau de contraction entre 500 et 1000. Le retrait de la contrainte d'équilibre permet d'obtenir des partitions de coût de coupe proche de 1.22. Cependant, dans ce cas, la balance de partitionnement se voit largement augmentée aux environs de 2.4. Ces résultats confirment que la méthode spectrale est initialement conçue pour réaliser un partitionnement non-contraint et que son utilisation dans un contexte contraint permet d'obtenir une partition de bonne qualité mais de coût de coupe nettement plus important.

Rappelons que la méthode spectrale est une méthode de recherche globale, il est donc intéressant de lui appliquer une méthode d'affinage afin d'optimiser localement la qualité de la partition. Pour cela, nous avons opté pour l'approche présentée en section 2.2.3, qui consiste à appliquer un affinage sur chaque niveau de la hiérarchie de modèles de la forme 2^i et à relâcher très légèrement la contrainte d'équilibre. Cette démarche permet de gagner jusqu'à 15% de coût de coupe, tout en respectant la contrainte balance < 1.05.

GGGP Vs Spectrale

La comparaison des résultats de partitionnement entre les méthodes GGGP et spectrale est réalisée à partir de trois catégories de graphes présentées en section 3.4.1. Chacun de ces graphes se compose de 10000 sommets et est partitionné pour un nombre de parties variant de 2 à 64. Le tableau 4.6 présente l'évolution du coût de coupe et de la balance pour chacun des partitionnements.

Méthode	**Grid 10000**					
	2	**4**	**8**	**16**	**32**	**64**
GGGP	15.469	31.305	74.296	103.09	160.51	218.17
balance	1.05	1.004	1.0352	1.0496	1.0688	1.0816
Spectrale	15.135	30.749	57.153	90.751	144.59	208.43
balance	1.006	1.0232	1.012	1.0496	1.0496	1.12
Méthode	**Tree 10000**					
	2	**4**	**8**	**16**	**32**	**64**
GGGP	1.8013	3.6767	7.3322	14.3333	27.2863	45.7317
balance	1.0386	1.048	1.0448	1.0448	1.0049	1.1008
Spectrale	9.4433	8.8045	13.701	27.8095	52.5477	54.3674
balance	1.0005	1.0172	1.0264	1.0448	1.0624	1.216
Méthode	**Linked 10000**					
	2	**4**	**8**	**16**	**32**	**64**
GGGP	20.277	41.281	94.663	158.79	279.13	447.26
balance	1.0028	1.0444	1.0496	1.0496	1.0496	1.088
Spectrale	24.879	44.973	82.67	165.06	284.12	436.23
balance	1.005	1.036	1.0496	1.0496	1.072	1.12

TABLEAU 4.6 – Comparaison des résultats obtenus par application des méthodes GGGP et spectrale

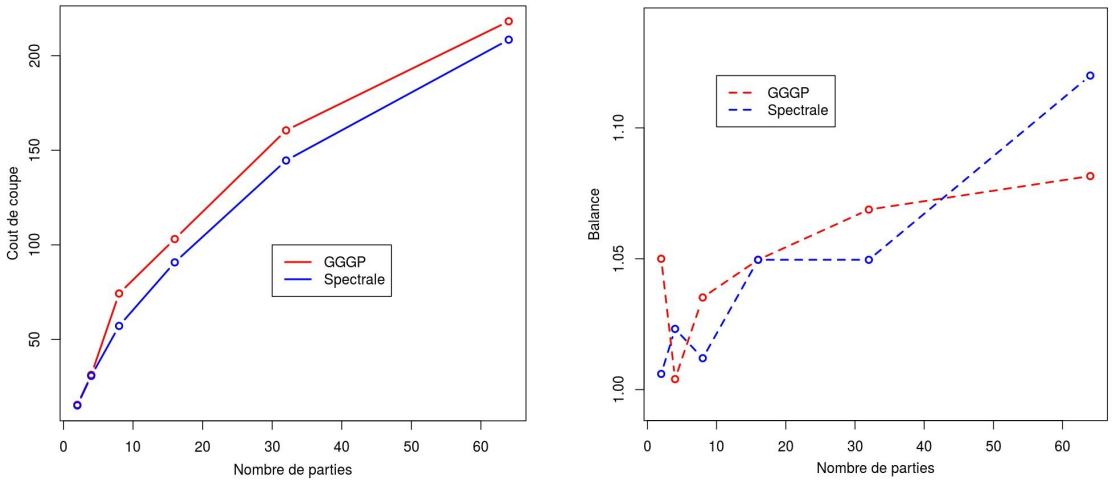


FIGURE 4.16 – Comparaison de l'évolution du coût de coupe et de la balance de partitionnement pour les méthodes GGGP et spectrale, sur un grid-graph

Pour le partitionnement d'un grid-graph de taille 10000, dont les résultats sont illustrés par les graphiques de la figure 4.16, nous pouvons constater que quelque soit le nombre de parties, la méthode spectrale offre toujours de meilleurs résultats que la méthode GGGP améliorée (à l'exception du 64-partitionnement où la balance de partitionnement est plus élevée sur le spectrale). L'observation seule de ces résultats nous porte à croire que la méthode spectrale est bien plus efficace que notre méthode GGGP. Or, en observant les résultats précédents, il semblerait que cette observation ne soit fondée que pour cette catégorie de graphes. En effet, les grid-graph ont une structure "simple" avec un nombre de connexions limitées (entre 2 et 4) et très régulières. La méthode spectrale repose sur la création de matrices qui reposent elles-mêmes sur ces connexions. Ce qui implique que les matrices obtenues sont parfaitement creuses, c'est à dire qu'elles se composent d'une majorité de 0. Nous pouvons donc émettre l'hypothèse que la méthode spectrale offre un meilleur partitionnement contraint lorsque sa matrice d'adjacence est creuse.

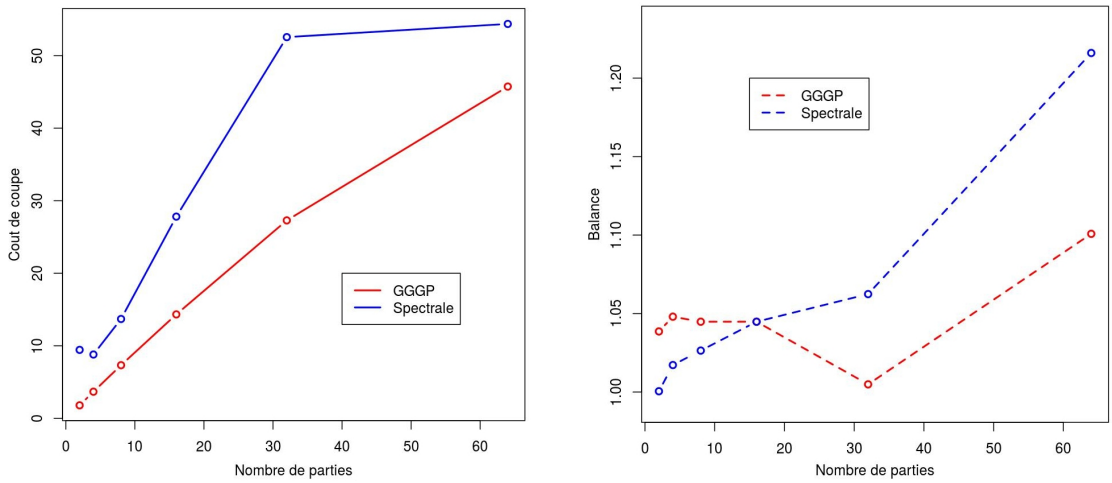


FIGURE 4.17 – Comparaison de l'évolution du coût de coupe et de la balance de partitionnement pour les méthodes GGGP et spectrale, sur un tree-graph

Pour le partitionnement d'un tree-graph de taille 10000, dont les résultats sont illustrés par les

graphiques de la figure 4.17, le constat est parfaitement opposé. En effet, dans ce cas, la méthode GGGP offre toujours un meilleur coût de coupe que la méthode spectrale, tout en garantissant le respect de la balance (à l'exception du 64-partitionnement, où la balance n'est pas respectée mais tout en restant plus faible que celle obtenue avec la méthode spectrale). Ces résultats tendent à confirmer la théorie des matrices creuses. En effet, le tree-graph étant un graphe "complexe", où le nombre de connexions est nettement plus important et beaucoup moins régulier, la matrice d'adjacence est donc beaucoup plus remplie que celle du grid-graph. N'oublions pas que la méthode GGGP repose sur un mécanisme de sélection locale, ce qui implique qu'elle est fortement influencée par la structure du graphe contrairement à la méthode spectrale. La présence de matrice creuse, ainsi qu'une structure parfaitement régulière pourraient permettre d'expliquer l'origine d'un partitionnement de moins bonne qualité pour le grid-graph.

Pour s'en convaincre, observons les résultats obtenus sur un linked-graph de taille 10000, dont les résultats sont illustrés par les graphiques de la figure 4.18. Dans ce cas, les résultats sont plus tempérés que ceux obtenus sur le tree-graph. Cependant, on peut constater que dans la majeure partie des cas, la méthode GGGP offre de meilleurs résultats que la méthode spectrale, à l'exception de la 8-partition et de la 64-partition. Pour cette dernière, le résultat est partiellement moins bon car même si le coût de coupe est plus élevé, la balance est de meilleure qualité. Pour ce type de graphe, la méthode spectrale manipule une matrice d'adjacence bien remplie, du fait qu'il existe un très grand nombre de connexions entre les sommets (pouvant allait jusqu'à 100 arrêtes pour certains sommets). Cependant, cette hyper-connectivité rend difficile le parcours du graphe par la méthode GGGP. Cette dernière explication peut permettre d'expliquer l'origine de l'écart relativement faible observé dans le tableau 4.6.

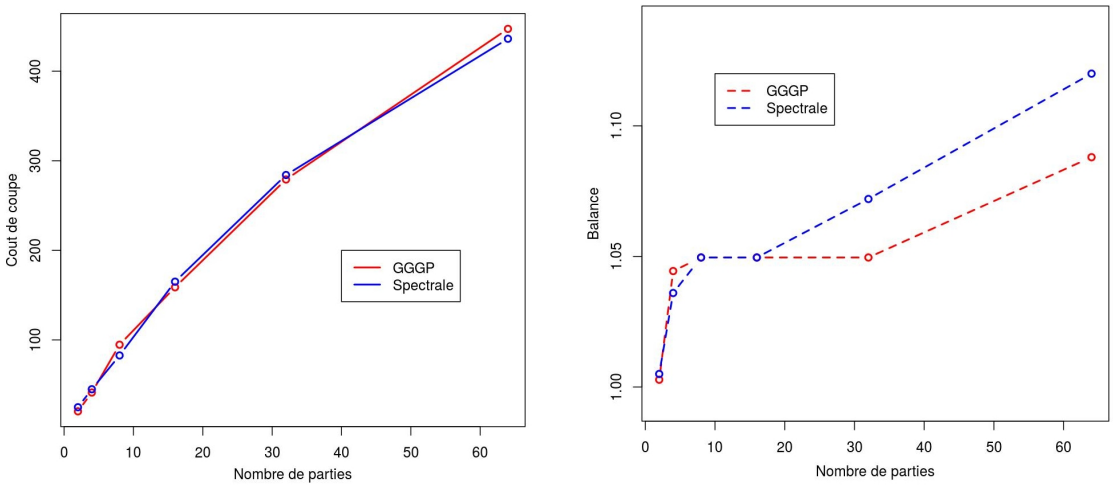


FIGURE 4.18 – Comparaison de l'évolution du coût de coupe et de la balance de partitionnement pour les méthodes GGGP et spectrale, sur un linked-graph

L'étude de ces résultats permet de mettre en avant le gain potentiel que peut offrir la méthode GGGP par rapport à la méthode spectrale s'il ne s'agit pas d'un graphe à structure simple et répétitive. L'écart de performance entre ces deux méthodes provient de la contrainte d'équilibre. En effet, sans cela la méthode spectrale offrirait très probablement un bien meilleur coût de coupe. Ceci étant lié à la nature de cette méthode.

4.4 Les effets de la restructuration de la hiérarchie de modèles sur les temps de simulation distribuée : le rôle du partitionnement

La distribution des modèles atomiques qui composent une simulation DEVS peuvent permettre, dans certains cas, d'obtenir un gain de temps considérable. Ce gain dépend en partie de la répartition des modèles atomiques au sein des nœuds de calcul, qui dépend elle même du partitionnement du graphe de modèles. Le partitionnement a donc un rôle prépondérant sur la distribution, d'où l'importance de bien cibler le problème et d'utiliser une méthode permettant d'y répondre au mieux. Nous verrons cependant que la qualité du partitionnement n'est pas toujours gage d'une distribution optimale. En effet, la dynamique des modèles atomiques a un impact très important sur la qualité de celle-ci. Lorsque les modèles atomiques sont synchrones, la qualité repose exclusivement sur le partitionnement. Cependant, plus les modèles atomiques tendent à être asynchrones, plus il est difficile de garantir l'exécution simultanée de modèles à une même date. Ceci étant lié à la taille des IMM. Le rôle de cette section est multiple. En effet, celle-ci présente l'évolution des speedups en fonction des méthodes de partitionnement de notre choix (GGGP et spectrale). Elle montre également l'impact qu'a la nature de la simulation (synchrone - asynchrone) sur l'évolution du speedup.

L'analyse des temps de simulation permet de bien comprendre l'origine du gain lié à la distribution. C'est pourquoi il est important de bien comprendre comment se compose le temps de simulation. Deux catégories de simulations sont à distinguer dans cette section :

- les simulations mono-threadé, où la simulation est prise en charge par un unique échéancier sur un seul noeud de calcul.

  Le temps obtenu pour ce type de simulation porte le nom de T_{mono}.

- les simulations distribuées sur n noeuds de calcul, où la simulation est mise en œuvre par n

  échéanciers de taille réduite. On parlera de temps distribué T_{dist}.

Le temps global observé est la somme des différents temps nécessaires au déroulement des différentes phases d'une simulation PDEVS. Dans le cas de la simulation mono-threadée, le temps obtenu correspond à la somme du temps de calcul des modèles (T_{cal}), du temps de traitement de l'unique échéancier (T_{mech}), du temps de routage des dates de réveil vers le coordinateur racine et des temps liés à la création des structures supposées négligeables ($\varepsilon \to 0$). Quelle que soit la catégorie, le coordinateur racine est localisé sur un noeud de calcul séparé. Dans le cas mono-threadé, deux noeuds sont donc utilisés et des échanges ont par conséquent lieu. Pour la simulation distribuée, le temps obtenu correspond à la somme du temps de calcul des modèles (T_{dcal}), du temps de traitement des échéanciers (T_{ech}), du temps lié au transfert des messages sur le réseau (T_{mess}) et du temps de routage des dates de réveil vers le coordinateur racine (T_{droot}). Pour ce dernier, les temps de création de structure sont également supposés négligeables.

$$T_{mono} = T_{cal} + T_{mech} + T_{root} + \varepsilon$$
$$T_{dist} = T_{dcal} + T_{ech} + T_{mess} + T_{droot} + \varepsilon$$

L'ensemble de ces temps peuvent être à l'origine de gain ou de perte de performance. En toute logique, pour un nombre de parties N, le temps de calcul en simulation distribuée est sensé être N fois plus petit, c'est à dire $T_{dcal} = \frac{T_{cal}}{N}$. D'autre part, le découpage de l'échéancier est lui à l'origine d'un gain de temps ($T_{mech} > T_{ech}$), mais il existe une perte liée à l'augmentation du nombre de messages transmis au coordinateur racine $T_{droot} = N * T_{root}$. De plus, l'utilisation du réseau est à l'origine d'une perte de temps liée au transit des messages T_{mess}. L'analyse de ces différents temps est nécessaire pour bien comprendre et bien interpréter les résultats qui vont être dans cette section.

4.4.1 Évolution du speedup pour des modèles synchrones, sans événement externe

Les résultats présentés dans cette section sont obtenus à partir d'un tree-graph de taille 10000, où la dynamique des modèles est celle des réservoirs, présentée en section 3.4.1. Afin d'estimer la performance de notre approche de restructuration de la hiérarchie par le biais d'une méthode de partitionnement optimisée, nous avons effectué une série de simulations où les modèles atomiques n'effectuent pas de transition externe, c'est à dire qu'aucun message n'est envoyé. Seuls les modèles effectuent un calcul lors de chaque transition interne, ce qui rend cette simulation purement synchrone puisque d'autre part, $ta = 1$. Les temps de simulation, présentés par la figure 4.19, sont obtenus pour un temps de calcul variant de 0.05 ms à 2 ms par modèle. L'intérêt de cette approche est de rendre négligeable les différents temps liés à la manipulation des échéanciers et à la construction des structures.

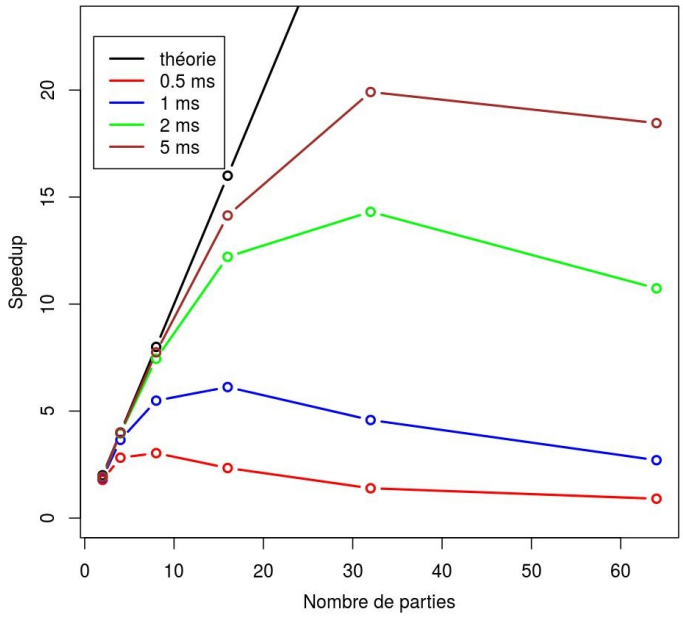


FIGURE 4.19 – Évolution du speedup d'une simulation synchrone sans échange de message en fonction du temps de calcul de chaque modèle atomique

La figure 4.19 nous montre que plus le temps de simulation est important, plus le speedup tend vers le speedup théorique, jusqu'à atteindre un speedup maximal à partir duquel il régresse. En effet, en observant l'évolution des courbes, nous pouvons constater qu'il existe un "front de propagation" du speedup maximum en fonction du nombre de parties. Pour un temps de calculs

très faible (0.05 ms), le meilleur speedup est atteint pour 8 parties, alors que pour 0.2 ms, il est atteint pour 16 parties et ainsi de suite.

En théorie, les modèles étant parfaitement équilibrés sur les nœuds de calcul et n'ayant aucune communication entre eux, nous serions sensés atteindre un speedup très proche du théorique. Cette limite du speedup maximum est liée au temps de calcul trop faible par rapport au temps de communication entre les coordinateurs et le coordinateur racine. En effet, même s'il n'existe pas de transfert de message entre les modèles, il est toujours nécessaire de faire remonter la prochaine date de réveil au coordinateur racine de façon à faire évoluer la simulation. Lors d'une simulation mono-threadée, pour une durée de simulation de 30 pas de temps, il y a 31 messages en transit entre le coordinateur et le coordinateur racine. Or, lors d'une distribution, on multiplie ce nombre de messages par le nombre d'échéanciers k. Comme le montre le tableau 4.7, plus le nombre de parties est important et plus le coût lié au transfert de messages entre les coordinateurs et le coordinateur racine est important. Ce qui explique les limites de performances du speedup. Lorsque le temps de calcul par modèle atomique est important, le coût lié au réseau devient négligeable. Ce qui explique l'augmentation de la qualité des speedup de la figure 4.19. En effet, en augmentant encore le temps de calcul, on peut espérer tendre vers le speedup théorique pour chaque nombre de parties. Pour une simulation où chaque modèle effectue tc ms de calcul, il est possible de déterminer le temps de calcul de la simulation Tcs (en seconde) à l'aide de la formule suivante :

$$Tcs = \frac{\text{nombre de transition interne } * tc}{1000}$$

Les k nœuds de calcul étant supposés équilibrés, le temps de calcul pour chaque nœud est d'environ $\frac{Tcs}{k}$. En incorporant cette information au tableau 4.7, on peut observer simplement l'origine de la perte du speedup.

Nombre de parties	Nombre de communications	Temps de communication	Temps de calcul pour 0.05 ms
1	31	0.372	15.5
2	62	0.744	7.75
4	124	1.488	3.875
8	248	2.976	1.9375
16	496	5.952	0.96875
32	992	11.904	0.484375
64	1984	23.808	0.2421875

TABLEAU 4.7 – Évolution, en parallèle, du coût de communication entre les coordinateurs et le coordinateurs racine, pour un temps de communication estimé par simulation de 0.012 seconde, et du temps de calcul en fonction du nombre de parties

Le temps de simulation inclut également le temps nécessaire à l'insertion dans le ou les échéanciers. Pour ce type de simulation, le temps de simulation s'obtient à l'aide de la formule suivante :

$$T_{simu} = I * Np(Tc + Te) + 31 * Np * Tr$$

Où,

- I est le nombre de transitions internes (310000 dans cet exemple)

- Tc est le temps de calcul en seconde (0.05/1000 ici)

- Te est le temps mis pour une insertion dans l'échéancier (estimé à 1.10^{-5} s)

- Tr est le temps de parcours d'un message vers le root (estimé à 0.012 s)

4.4.2 Évolution des modèles synchrones, aux événements externes asynchrones

Contrairement aux dynamiques des modèles de la sous section précédente, celles présentées ici échangent des événements à chaque transition externe. La simulation est réalisée à partir d'un tree-graph de taille 10000, où chaque modèle atomique possède une dynamique de remplissage de réservoirs à changement de vitesse et effectue un temps de calcul variant de 0.5 à 5 ms par transition interne. Les modèles possèdent tous un t_a égal à 1, par conséquent, ils sont tous synchrones. Au cours de la simulation, 196542 transitions internes et 84238 transitions externes sont réalisées. Au vu de ces paramètres de simulation et des observations faites en section 4.4.1, il semble naturel de s'attendre à obtenir des speedups proches de ceux observés dans la figure 4.19.

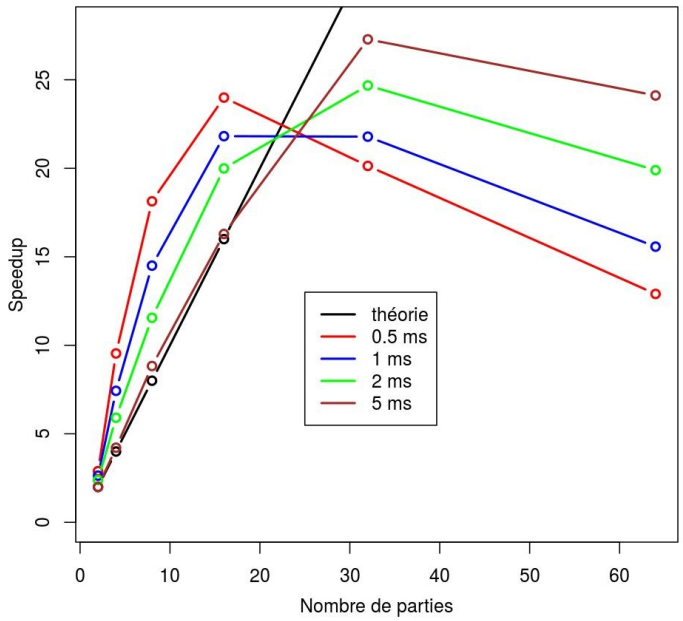


FIGURE 4.20 – Évolution du speedup d'une simulation synchrone avec échanges de messages, incluant le gain de temps lié au découpage de l'échéancier, en fonction du temps de calcul de chaque modèle atomique

La figure 4.20 présente les résultats de simulation distribuée obtenus pour ce type de simulation. Au premier coup d'œil, ces résultats semblent incohérents voire totalement faux. En effet, pour une distribution à n noeuds de calcul, il est impossible d'obtenir un speedup supérieur à n. Cependant, nous pouvons observer que lorsque le temps de calcul augmente, les speedups tendent de plus en plus vers la théorie. En effet, les courbes tendent à se rapprocher du speedup théorique (courbe noire). Rappelons qu'il est possible de décomposer le temps de simulation distribuée comme suit :

$$T_{dist} = T_{dcal} + T_{ech} + T_{mess} + T_{droot} + \varepsilon$$

Or, plus le temps de calcul augmente, plus le temps lié au routage des événements (réseau + coordinateur racine) et à la manipulation des échéanciers devient négligeable. Le fait que l'augmentation du temps de calcul par modèle fasse tendre les résultats vers le speedup théorique tend à montrer que ces résultats sont parfaitement cohérents. Mais comment expliquer l'origine du gain additionnel ? Au vu des observations faites dans la partie précédente, il semblerait que ce gain provienne à la fois du découpage de l'échéancier et de la manipulation des événements externes.

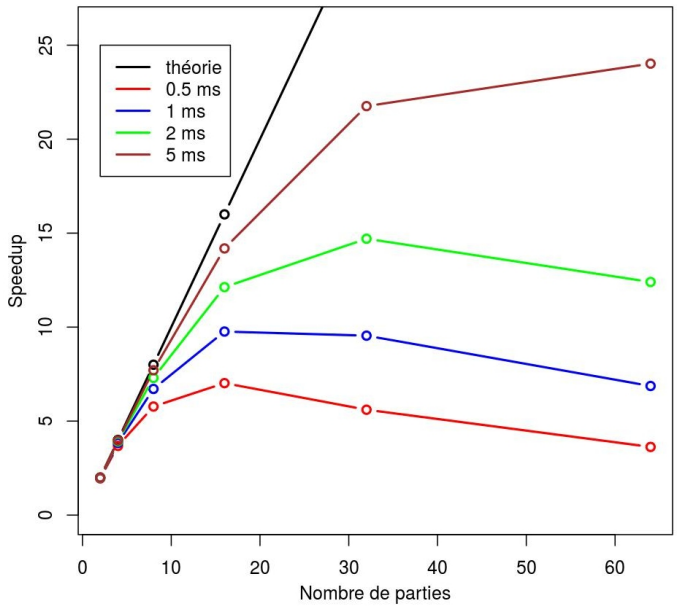


FIGURE 4.21 – Évolution du speedup d'une simulation synchrone avec échanges de messages, n'incluant pas le gain de temps lié au découpage de l'échéancier, en fonction du temps de calcul de chaque modèle atomique

Contrairement à la simulation de la partie 4.4.1, celle-ci comporte des événements externes. Or, à chaque transition externe, une nouvelle date est introduite dans le ou les échéanciers. Cette insertion peut être réalisée à l'aide de deux opérateurs sur la structure de données utilisées - heap : *increase* et *decrease*. En temps normal, ces deux opérateurs ont une complexité respective de $O(1)$ et $O(log(N))$, où N correspond à la taille de l'échéancier. Or, comme les événements sont rangés à l'inverse du fonctionnement normal d'un heap (structure d'accueil d'un échéancier), on peut en déduire que les complexités sont inversées : *increase* ($O(log(N))$) et *decrease* ($O(1)$). Or, dans cette simulation, il y a 20315 appels à la fonction increase et 280842 pour decrease. La complexité de chaque insertion d'événements externes fait qu'il est moins coûteux d'insérer un événement dans un échéancier de taille N/k que sur un échéancier de taille N. Cependant, cette seule insertion ne suffit pas à expliquer la totalité du gain observée sur la figure 4.20. Une autre structure est appelée à chaque occurrence d'un événement externe : une multimap. Cette structure est utilisée pour le stockage des connexions entre modèles. Elle met en relation des ports de sortie des modèles aux ports d'entrée des autres modèles. Les sources ou les destinations peuvent être le modèle couplé lui-même. Lors de l'émission d'un événement externe, nous devons faire appel à une recherche au sein de cette structure. Or, cette opération possède une complexité de l'ordre de $O(log(N))$. Une nouvelle fois, un gain apparaît lors du passage du modèle complet à des sous-

graphes dont le nombre de connexions est inférieur. Dans le cas de nos méthodes, la plupart du temps le facteur de réduction est de l'ordre du nombre de parties. Pour 2 parties, on passe donc d'environ 24000 connexions à 12000.

L'accumulation de tous les gains, provenant du découpage de l'échéancier et du graphe, permet finalement d'expliquer l'origine des résultats présentés par la figure 4.21. En effet, au vu des structures manipulées, il est beaucoup moins coûteux de manipuler k échéanciers de taille N/k et k graphes de taille M/k plutôt qu'un unique échéancier de taille N et un unique graphe avec M connexions. Cette théorie semble être confirmée par les observations et les résultats présentés en section 4.2.1. En supprimant le gain de temps lié au découpage des échéanciers, il est possible d'observer uniquement le temps de simulation $T_{dist} = T_{dcalc} + T_{mess} + T_{root}$. La figure 4.21 estime l'évolution des speedups, sans inclure le gain de temps lié au découpage des échéanciers, pour un temps de calcul par modèle variant de 0.5 à 5 ms.

La suppression des temps liés à la manipulation des échéanciers permet d'obtenir des courbes en accord avec la réalité. En effet, les résultats présentés sont obtenus pour un partitionnement à partir de la méthode GGGP. Les communications entre les nœuds de calcul sont donc supposées minimales. Les baisses de performances constatées sont donc liées au routage des messages sur le réseau et entre les coordinateurs et le coordinateur racine. Il est bon de rappeler que ces résultats sont une estimation globale de ce que l'on pourrait espérer si le gain lié au découpage de l'échéancier n'était pas si important.

4.4.3 Influence de la méthode de partitionnement sur l'évolution du speedup

La distribution des modèles atomiques au sein des nœuds de calcul doit être faite de façon à respecter un équilibre de charge de calculs et une minimisation de transfert de messages entre chaque nœud. En effet, pour garantir une simulation optimale en temps dans un contexte synchrone, il est nécessaire d'exécuter un maximum de modèles atomiques en simultané. Cependant, pour éviter les pertes liées à l'utilisation du réseau, il est important de minimiser l'envoi de messages entre modèles de différents nœuds de calcul. Cette répartition optimale des modèles atomiques au sein des nœuds de calcul repose exclusivement sur le partitionnement du graphe de modèle. En effet, nous avons vu en partie 2.1.2 qu'il est important de bien formaliser le problème de partitionnement de façon à ce qu'il reflète au mieux notre problématique de distribution optimale. Pour garantir l'obtention d'une distribution optimale, nous sommes partis du postula que la partition générée doit respecter les deux contraintes suivantes :

- équilibre de charge entre les parties (contrainte forte)
- minimisation du coût de coupe (contrainte faible)

Le rôle de cette section est de comparer les speedups des simulations obtenues par application de méthodes intelligentes à ceux obtenus par application de méthodes naïves. En effet, sans la connaissance liée au graphe de modèle, il est difficile, voire impossible de répartir équitablement les modèles au sein des nœuds de calcul. Afin de simuler une répartition sans cette connaissance, nous avons mis en place deux types de partitionnement :

- un partitionnement aléatoire (rand), où chaque partie ne possède pas le même poids et où le nombre de communications est très élevé.

- un partitionnement aléatoire équilibré (rand équilibré), où chaque partie possède le même poids mais où le nombre de communications est très élevé.

Il est possible de voir cette seconde méthode comme étant une approche que pourrait avoir un modélisateur. En effet, celui-ci a une connaissance globale du modèle lui permettant d'anticiper un certain équilibre lors de la répartition des modèles atomiques. Mais il n'est pas en mesure de prédire la quantité de communication que cette répartition peut engendrer, ou tout du moins sans une grande précision.

Les résultats présentés par la figure 4.22 sont à comparer qualitativement. En effet, à l'image de la partie 4.4.2, les speedups présentés ici sont largement supérieurs aux speedups théoriques. Les raisons liées à ces résultats sont les mêmes que celles présentées dans la section précédente (gain important lié à la réduction de la taille des échéanciers et des structures manipulées lors du traitement des événements externes). Dans cette sous-section, la simulation est réalisée à partir d'un tree-graph de taille 10000, où chaque modèle possède une dynamique de remplissage de réservoir à changement de vitesse et un temps de calcul de 0.5 ms. Ce qui implique que les événements externes sont activés et qu'il y a transfert de messages entre les modèles atomiques des différents nœuds de calcul. Les résultats présentés par la figure 4.22 sont obtenus par application des quatre méthodes de partitionnement suivantes : gggp, rand, rand équilibré et spectrale.

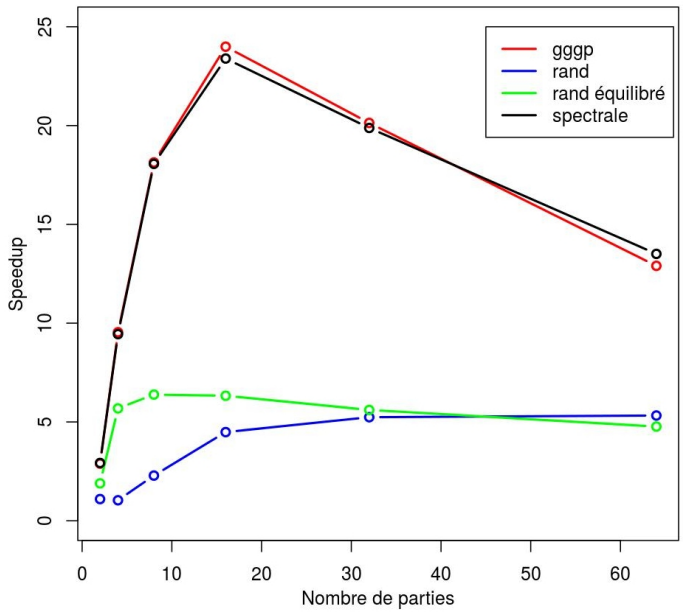


FIGURE 4.22 – Comparaison de l'évolution du speedup en fonction de la méthode de partitionnement choisie

Nous pouvons constater que les résultats obtenus par application des méthodes GGGP et spectrale sont très proches, même si GGGP offre un très léger gain dans ce cas. De plus, en reprenant ce qui a été vu en section 4.4.2, c'est à dire en supprimant le gain lié au découpage des échéanciers, nous pouvons constater que les speedups tendent vers la théorie. Rappelons qu'au vu du

petit temps de calcul de chaque modèles le temps lié au routage des évènements vers le coordinateur racine devient plus important que le temps de calcul à partir d'un certain nombre de parties (voir 4.4.1). De plus, nous pouvons constater que, même si la méthode rand équilibré offre globalement de meilleurs résultats que rand, elles restent toutes deux bien moins performantes que les méthodes spectrale et GGGP. Tentons d'expliquer ce phénomène à l'aide des caractéristiques de chaque partitionnement. La figure 4.23 présente, d'une part, l'évolution des coûts de coupe pour chaque méthode et, d'autre part, l'évolution des balances de partitionnement.

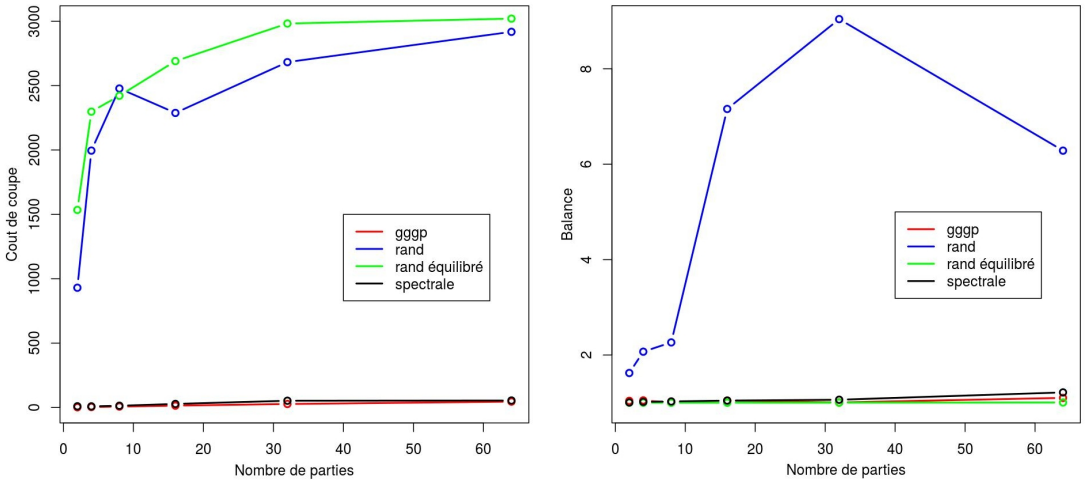


FIGURE 4.23 – Évolution des coûts de coupe et des balances de partitionnement pour chacune des méthodes

Contrairement aux méthodes GGGP et spectrale, les méthodes à base d'aléatoire, rand et rand équilibré, possèdent un coût de coupe nettement plus élevé (entre 100 et 300 fois). Ce qui se traduit par une explosion des temps de communication via le réseau. De plus, si on compare les balances de partitionnement, nous pouvons constater que seule la méthode rand possède une partition déséquilibrée (rand équi étant la méthode la mieux équilibrée). Le déséquilibre de charge, ainsi que le nombre de communications exorbitant explique la faible performance de la méthode rand. Au vu de ces observations, nous nous serions attendus à obtenir une bien meilleure performance de la part de la méthode rand équilibré. Cependant, étant donné le très grand nombre de messages en transit sur le réseau, le gain engendré par l'équilibre des modèles est perdu à cause des temps de communications entre les nœuds de calcul. Cet exemple met en lumière l'importance du respect des contraintes lors du partitionnement. En effet, pour garantir une simulation distribuée optimale (en mode synchrone), il est nécessaire d'avoir une répartition homogène des modèles et de minimiser le plus possible le nombre de communications entre les nœuds de calcul.

La figure 4.24 présente l'évolution d'une simulation synchrone, où chaque modèle atomique possède un temps de calculs de 5 ms et où le nombre de transitions internes est cinq fois plus petit que le nombre de transitions externes. Cette spécificité fait que les modèles atomiques sont hyper-communicants. Par conséquent, les coûts de transfert liés au réseau se voient très largement augmentés. En vue du nombre de transitions internes, nous sommes en présence d'une simulation où le temps de calculs est moins important que le coût de communication (même avec 5 ms de calculs par modèle atomique). Les résultats de la figure 4.24 tendent à confirmer ces propos.

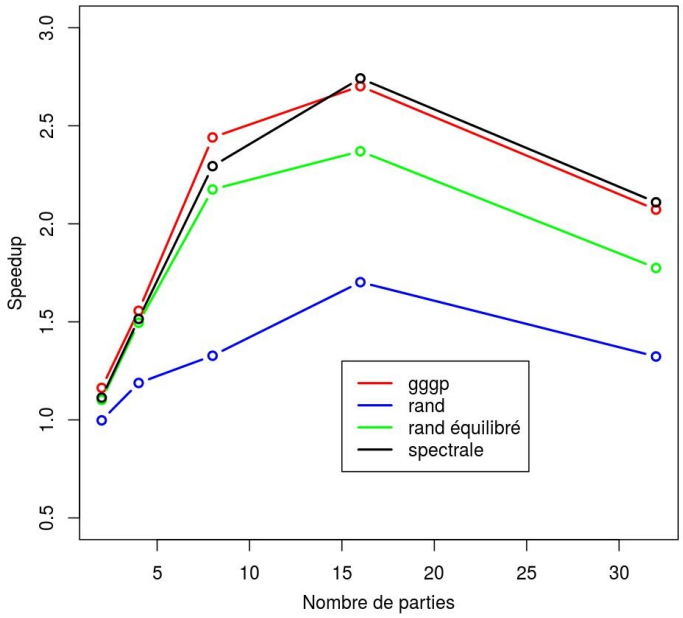


FIGURE 4.24 – Comparaison de l'évolution du speedup en fonction de la méthode de partitionnement choisie pour un linked-graph de petite taille, se composant de modèle synchrones hyper-communicants

En effet, en comparaison des résultats des parties 4.4.1 et 4.4.2, les résultats semblent beaucoup moins bons. Cependant, ceci n'est pas lié au partitionnement mais exclusivement au coût de communication qui est plus important que le gain de temps lié à la distribution des modèles. Ces coûts sont à l'origine de cette baisse de performance.

4.4.4 Évolution du speedup pour des modèles asynchrones

Lorsque les modèles atomiques sont synchrones (tous les t_a sont égaux), nous avons pu constater une très grande efficacité de notre stratégie de restructuration de la hiérarchie de modèles à partir de nos méthodes de partitionnement contraint. En effet, lorsque le temps de calcul n'est pas négligeable face au coût de transfert de messages, notre partitionnement permet de garantir un speedup très proche du speedup théorique. Cet équilibre dépend du nombre de transitions internes, du nombre de transitions externes mais aussi de la qualité du partitionnement. Cette qualité provient, en partie, de la minimisation des coûts de transferts de messages entre les nœuds de calculs. Cependant, ces résultats ne sont pas généralisables à toutes les simulations DEVS. En effet, nous allons voir dans cette partie que lorsque les modèles ont des comportements totalement différents (t_a différents), il est très difficile de réaliser une distribution garantissant une exécution simultanée d'un maximum de modèles.

Les résultats présentés par la figure 4.25 sont obtenus pour un linked-graph de taille 300, où chaque modèle possède un temps de calcul nul et où chaque modèle possède son propre t_a tiré aléatoirement. Au cours de la simulation, 14628 transitions externes sont réalisées. Cette expérience est réalisée sur un graphe de petite taille dans le but de rendre négligeable le gain de temps lié au découpage des échéanciers. La figure 4.25 présente l'évolution des speedups pour chaque méthode de partitionnement. Les résultats catastrophiques de cette simulation s'expliquent de deux façons. La première vient du fait que les modèles sont asynchrones, ce qui implique que chaque modèle

effectue son exécution indépendamment des autres modèles. En effet, comme chaque modèle possède un t_a tiré aléatoirement, il est rare de trouver plusieurs modèles dont la transition interne a lieu au même moment. De ce fait, même si la partition générée par GGGP est théoriquement de très bonne qualité, il est difficile d'obtenir des *IMM* dont la taille diffère de un. Ce qui se traduit par un speedup proche de 1 (au maximum 2 dans le meilleur des cas). La distribution avec une politique pessimiste n'a aucun intérêt dans ce cas, car les modèles sont exécutés les uns après les autres. De plus, la figure 4.25 tend à montrer que, dans ce cas, la distribution tend à empirer les résultats à cause du coût lié à l'utilisation du réseau et au routage des événements vers le coordinateurs racine. En effet, dans cet exemple, il y a deux fois plus de transitions externes que de transitions internes, les modèles sont hyper-communicants. De plus, les temps de calculs étant nuls, nous ne pouvons observer que le coût lié au routage des événements et à l'utilisation du réseaux, ce qui explique totalement l'origine des speedups inférieurs à 1.

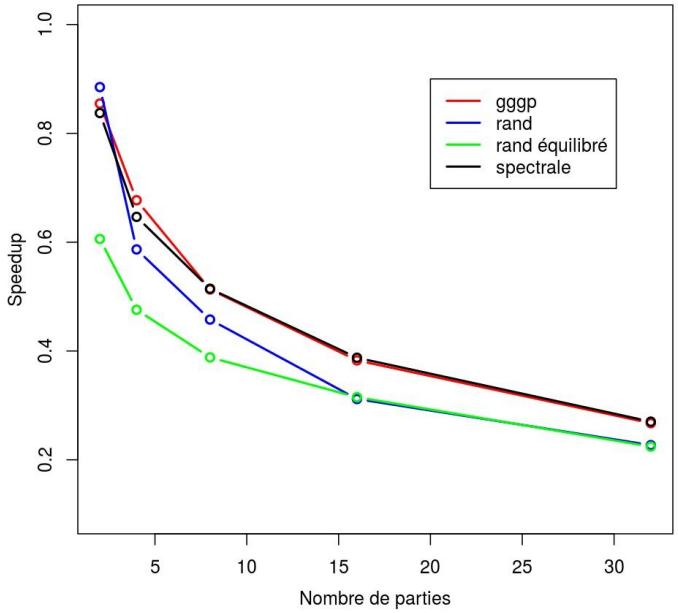


FIGURE 4.25 – Évolution des speedups pour une simulation totalement asynchrone, où le nombre de transitions externes est deux fois supérieur au nombre de transitions internes et pour un temps de calculs nul. Comparaison des différentes méthodes de partitionnement

Ces résultats ont tout de même l'avantage de confirmer la supériorité des méthodes de partitionnement intelligentes (GGGP et spectrale). En effet, même dans un cas catastrophique, les résultats obtenus par ces méthodes sont toujours meilleurs que ceux obtenus par les méthodes aléatoires. La différence de résultats que l'on peut observer dans cet exemple est uniquement liée au surcoût engendré par l'utilisation du réseau. En effet, les méthodes aléatoires véhiculent entre 10 et 100 fois plus de messages que les méthodes intelligentes. L'augmentation des temps de calcul à 5 ms pour une telle simulation n'a pour effet que de masquer, en partie, les pertes liées au routage des messages et à l'utilisation du réseau. En effet, comme le montre la figure 4.26, le speedup n'excède pas les 1.8 et décroit considérablement au-delà de quatre parties. Ces résultats montrent qu'à cause des t_a aléatoires, il est impossible d'exécuter simultanément plus de deux modèles sur une longue durée dans ce cas.

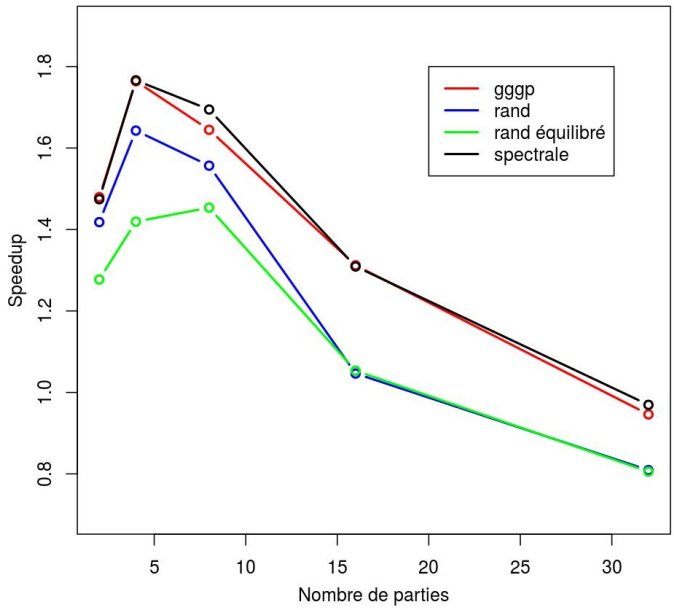


FIGURE 4.26 – Évolution des speedups pour une simulation totalement asynchrone, où le nombre de transitions externes est deux fois supérieur au nombre de transitions internes et pour un temps de calculs de 5 ms par modèle. Comparaison des différentes méthodes de partitionnement

Conclusion

La distribution des modèles atomiques au sein de plusieurs nœuds de calcul représente une piste d'optimisation des temps de simulation. En effet, lorsque la parallélisation n'est pas suffisante en terme de mémoire, cette dernière offre un bon compromis à condition de bien distribuer les modèles. Contrairement à la parallélisation, les coûts de communication entre les nœuds de calcul n'est pas négligeable. Il est donc très important de minimiser le transfert d'événements via le réseau. Cette perspective met en avant la nécessité d'utiliser une méthode de partitionnement garantissant un équilibre de charge et une minimisation du coût de transfert. À l'image des résultats présentés en section 4.4, à équilibrage équivalent, il a été mis en avant que le coût lié à l'utilisation répétée du réseau engendre une perte de temps non négligeable. Cependant, ces résultats n'ont pas permis de mettre en avant toute l'importance de la pondération du graphe de modèles, par le biais de l'apprentissage des dynamiques à l'aide de chaînes de Markov cachées. En effet, les modèles du graphe se composent d'une unique dynamique offrant une faible variation de la fréquence d'émission d'événements. Par conséquent, le coût de coupe obtenu à l'aide d'un graphe non pondéré n'est que très peu éloigné de celui obtenu à partir d'un graphe pondéré à l'aide d'une procédure d'apprentissage minimaliste. Cette remarque est corroborée par les résultats présentés dans la partie 4.3.1, même si le partitionnement est meilleur sur un graphe pondéré. L'écart entre les deux résultats n'est pas suffisamment important pour observer une évolution conséquente du speedup (speedup à $\pm$ 0.2). C'est la raison pour laquelle, aucune comparaison n'est effectuée entre distribution obtenue à partir d'un graphe pondéré et un graphe non pondéré. Cependant, cet exemple de dynamique à variation faible ne suffit pas à réfuter l'utilisation de la procédure d'apprentissage. Il semble naturel d'affirmer toute l'importance de cette procédure pour les graphes à

dynamique variable en fonction de la localisation des modèles. En effet, un graphe avec une grande fluctuation de fréquences d'émission d'événements aura beaucoup plus de chance de fournir un partitionnement médiocre si celui-ci est réalisé sans prendre en compte la pondération des arcs.

Cette section met en avant tout l'intérêt apporté par la restructuration de la hiérarchie, par le biais d'une méthode de partitionnement, pour la réduction des temps de simulation où les modèles sont synchrones. Dans ce cas, pour un temps de calcul non négligeable, le speedup de simulation tend vers le speedup théorique. C'est d'autant plus vrai que le temps de calcul est important. Dans ce cas, les temps de communication n'impactent que peu, voire pas du tout, le temps de simulation. Il a également était mis en avant, que même lorsque les modèles sont totalement asynchrones, la restructuration de la hiérarchie permet de réaliser du gain, même si celui-ci est loin de tendre vers l'optimum. Cependant, sachant que pour ce type de simulation, les t_a sont réels et en majorité différents, nous pouvons être satisfaits des résultats présentés en section 4.4.4. En effet, car à une même date t, les ensemble *IMM* ne possèdent pas tous la même taille et leurs tailles sont très loin d'être optimales vis à vis de l'architecture matérielle. Par conséquent, il est rare de pouvoir exécuter plusieurs modèles en parallèle. Cette perte de performance est totalement indépendante du paramétrage de la méthode de partitionnement. En effet, dans certains cas extrêmes, il est parfaitement impossible d'exécuter plus d'un modèle à la fois. Cependant, il existe un moyen de réduire l'impact des t_a : la simulation optimiste. La simulation pessimiste empêche l'évolution d'une partie de la simulation au risque d'être stoppée par manque d'informations. Tandis que le passage à la simulation optimiste autorise ce type de comportement au risque de devoir effectuer un rolback (retour en arrière). La stratégie du passage à la simulation optimiste est gagnante à condition de ne pas effectuer d'importants retours en arrière, et à de très faibles occasions. Pour limiter ces potentiels rolbacks, il est possible de définir une fenêtre temporelle (lookahead) au cours de laquelle on autorise l'évolution des modèles. Ce passage à l'optimiste représente une perspective sérieuse pour l'optimisation des simulations distribuées asynchrones.

Les résultats de la partie 4.4 mettent également en avant l'importance d'utiliser une méthode de partitionnement respectant au mieux les contraintes liées à la distribution des modèles atomiques. En effet, quel que soit le cas, la restructuration obtenue à partir d'une méthode GGGP, ou spectrale, est toujours meilleure que celle obtenue à partir d'une méthode aléatoire, même équilibrée en charge. Ceci montre bien l'importance de minimiser le nombre d'événements en transit sur le réseau, pour ne pas affecter le gain lié à l'équilibre. Le peu de différence observée entre les méthodes GGGP et spectrale, vient du fait que les simulations utilisées lors de la phase de tests possèdent des dynamiques très peu variables.

Chapitre 5

Application à la gestion durable du Mildiou de la pomme de terre

La culture de la pomme de terre est l'une des principales cultures alimentaires mondiales avec une production d'environ 368.1 millions de tonnes en 2013 à travers le monde et une production nationale d'environ 5.2 millions de tonnes. La pomme de terre représente un important enjeu économique et sanitaire. En effet, de part sa structure, la pomme de terre est potentiellement exposée, tout au long de son cycle de production, à un très grand nombre de bio-agresseurs tels que les insectes, les bactéries , les virus, etc ... mais son principal ennemi reste le mildiou *P. infestans*. La lutte contre ce dernier est l'un des enjeux majeurs pour cette culture. Depuis plusieurs années, la France a mis en place de nombreuses protections phytosanitaires, auxquelles est associé un haut niveau de technologie, pour garantir le contrôle et la propagation du mildiou *P. infestans*. Cependant, les effets nocifs de l'emploi des pesticides sur la santé des utilisateurs et sur l'environnement amènent aujourd'hui à les utiliser d'une façon plus raisonnable. En effet, il s'agit à la fois de réduire l'usage de ces produits et de limiter l'impact de ceux qui resteront indispensables pour protéger les cultures des maladies. Il semble essentiel de proposer des stratégies de gestion durable de la maladie, moins dépendantes de l'utilisation des pesticides.

Cette limitation de l'usage des pesticides passe par la mise en place d'outils d'aide à la décision basés sur des modèles, des capteurs (stations météo par exemple), de la simulation, permettant d'évaluer le risque de contamination des pommes de terre par le mildiou *P. infestant*. En effet, la création d'un tel outil ouvrirait la voie de la limitation de l'usage préventif des pesticides, comme c'est le cas actuellement. Le mildiou *P. infestant* est une maladie très virulente au temps d'incubation extrêmement rapide. En effet, lorsque la maladie est visible par l'homme cela implique, dans la quasi-totalité des cas, que la récolte est déjà gravement compromise. Ce qui explique le souci de sécurité des agriculteurs. En effet, au vu du prix des pesticides, il est nettement plus économique pour un agriculteur d'en faire un usage préventif plutôt que de risquer de perdre sa récolte. L'approche de plusieurs acteurs de la filière pour limiter l'usage préventif des pesticides est de fournir un outil d'aide à la décision permettant d'informer l'agriculteur sur un éventuel risque de contamination en fonction des conditions climatiques et des informations liées à l'état des cultures. Les travaux présentés dans ce chapitre ont donc pour objectif d'apporter des éléments potentiellement

intéressants pour la construction de tels outils.

Actuellement, les outils de modélisation de la prolifération du mildiou *P. infestant* sont limités spatialement et reposent sur des modèles vieillissants ne prenant pas en compte tous les facteurs de risques (tel que le vent). À partir des travaux de thèse de Toky Fanambinana Rakotonindraina [Rak12] financée par les divers organismes autour de la pomme de terre regroupés au sein de l'ACVNPT, de nos collaborations avec l'INRA et le monde agricole principalement localisé sur le Littoral Nord de France, nous proposons dans cette thèse un modèle mildiou revisité permettant une utilisation de ce dernier sur des échelles spatiales nettement plus importantes.

5.1 Du développement de la plante aux effets néfastes du mildiou

Les enjeux sociaux économiques liés à la pomme de terre, à l'échelle de la région, du pays et du monde, sont tels que de nombreuses recherches ont été réalisées, tant sur le plan agronomique qu'informatique. C'est notamment le cas pour la thèse de Toky F. Rakotonindraina, qui allie des recherches agronomiques et la mise en place de modèles DEVS à l'aide de la plate-forme RECORD (REnovation et COoRDination de la modélisation de cultures pour la gestion des agro-systèmes) [BCG+13] permettant la simulation de la propagation du mildiou au sein d'une parcelle. L'ensemble des informations présentées dans cette section sont soit extraites, soit en rapport avec la thèse de Tokiy [Rak12].

5.1.1 Qu'est ce qu'une pomme de terre ?

La pomme de terre est un tubercule comestible appartenant à la famille des solanacées. Le terme désigne également la plante elle-même, illustrée par la figure 5.1, qui a la particularité de proliférer dans la plupart des sols.

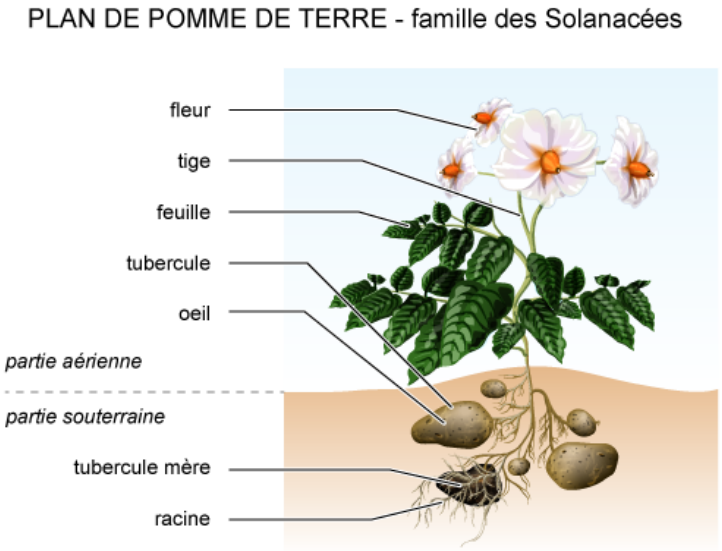


FIGURE 5.1 – Schéma détaillé d'une pomme de terre (plante) [source : *http ://aragonpantin.fr*]

La pomme de terre est une plante annuelle dont le cycle se compose en trois phases :

- la première correspond à la croissance des germes à la sortie du repos végétatif d'un tubercule. Ces germes se transforment en tiges feuillées et en rameaux dans la partie aérienne, et en stolons dans la partie souterraine.

- la seconde débute par la phase de tubérisation lorsque les stolons arrêtent de s'allonger. Durant cette période, les ébauches de tubercules grossissent grâce au transfert des substances de réserve synthétisées au niveau du feuillage vers la partie souterraine de la plante. Lorsque le système aérien est totalement desséché (cf sénescence), les tubercules cessent de croître du fait qu'ils sont arrivés à maturation, et sont donc prêts à être récoltés.

- après la récolte, les tubercules entrent dans un état de repos végétatif durant lequel leurs bourgeons sont incapables, même en conditions favorables, de croître pour germer. Quand cet état de dormance est levé, les tubercules vont germer : c'est la germination qui annonce le début d'un autre cycle végétatif.

5.1.2 Qu'est ce que le mildiou *P. infestant* ?

Le mildiou de la pomme de terre, causé par *Phytophthora infestans* de Bary, est l'une des maladies les plus redoutables de la culture de pomme de terre. En effet, la maladie apparaît dans une parcelle sous forme de foyers isolés, à partir desquels elle peut se répandre rapidement et aboutir à une destruction totale de la végétation en moins de deux semaines. Le mildiou se propage sous forme de spores pouvant attaquer tous les organes de la plante, à l'exception des racines. Cette maladie n'est généralement traitée qu'en préventif principalement. En effet, une fois qu'une parcelle est atteinte, il est difficile voire presque impossible d'empêcher sa prolifération. L'apparition du mildiou à différents stades de croissance de la plante n'a pas le même effet sur le rendement et sur la qualité. En effet, une apparition précoce provoque surtout une diminution de la photosynthèse, liée à la nécrose des feuilles, pendant la phase de tubérisation, pouvant être à l'origine de la mort des tubercules ou au minimum, au ralentissement de leur croissance. Tandis qu'une apparition tardive provoque une baisse de la qualité des tubercules, notamment lorsque la contamination a lieu par le sol (voir l'illustration de la figure 5.2).

FIGURE 5.2 – Exemple de nécrose liée à l'apparition précoce du mildiou (à gauche) [source : *http ://www.agro.basf.fr/*] et présentation d'une vue externe et interne de l'effet du mildiou sur un tubercule lors d'une apparition tardive (à droite) [source : *https ://www6.inra.fr*]

Cycle de développement et prolifération

Un cycle de la maladie (monocycle) correspond à la période qui s'écoule entre deux générations de spores, de l'infection à la production d'une nouvelle génération de spores. Ce cycle se décompose lui-même en plusieurs étapes :

- la période d'incubation correspondant au temps écoulé entre l'infection et l'apparition des premiers symptômes

- la période de latence correspondant au temps écoulé entre l'infection et la production de nouvelles spores

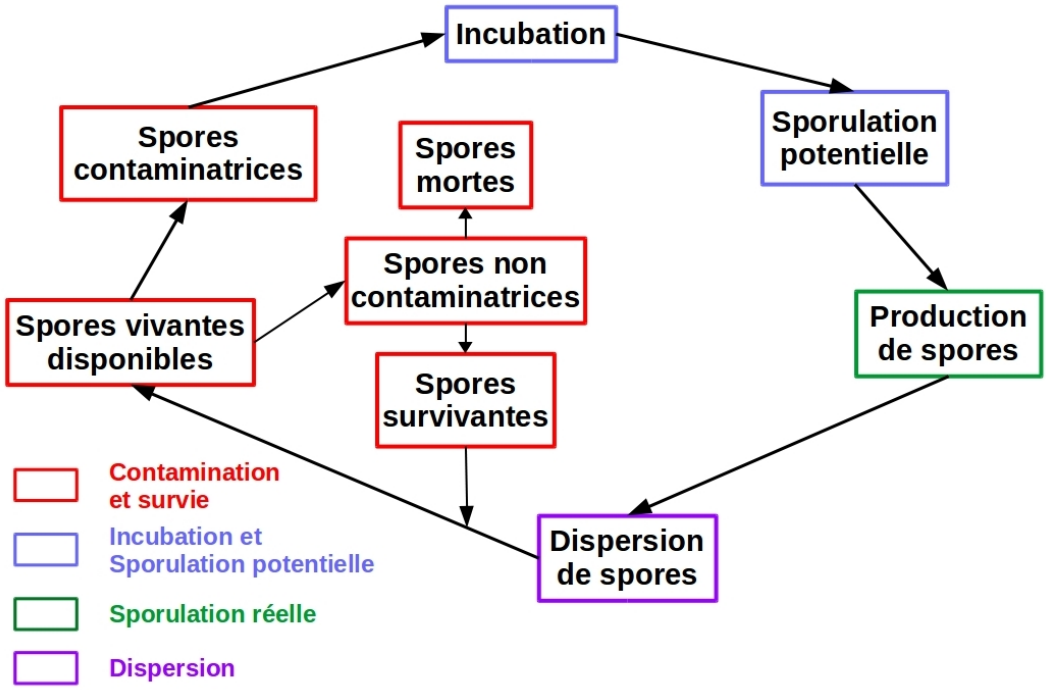


FIGURE 5.3 – Cycle de génération du mildiou *P. infestant*

Il existe deux catégories de spores : les asexuées et les sexuées. La dispersion des spores asexuées par le vent, ou la pluie, forme le point de départ d'une épidémie de mildiou. La courte durée d'un monocycle, souvent amplifiée par une production importante de spores asexuées, fait que l'épidémie de mildiou peut être très rapide : elle est qualifiée d'explosive. La dispersion locale des spores asexuées permet l'apparition de foyers de mildiou à l'échelle de la parcelle. Celle-ci est favorisée par l'effet *"splashing"*, correspondant au déplacement des spores présentes sur les feuilles ou le sol à l'aide des éclaboussures liées à la chute des eaux de pluie. Cette dispersion est véritablement très localisée, contrairement au principal agent de dispersion qui n'est autre que le vent. L'action de celui-ci, favorise l'extension de la maladie au-delà de la parcelle. L'épidémie peut être qualifiée de régionale, car les spores de contamination peuvent être transportées jusqu'à 11 km de distance des foyers primaires.

Facteurs de contamination

Le mildiou P. infestans se caractérise par un cycle majoritairement aérien, illustré par la figure 5.3. En effet, un ensemble de filaments (le mycélium) se développe dans les organes aériens et les tubercules pour produire, à leur surface, les structures végétales permettant d'accueillir et de produire des spores (sporanges). Ces sporanges peuvent être dispersées par voie aérienne ou par l'eau afin d'atteindre le feuillage des plantes voisines. Il faut un taux d'humidité relative supérieur à 90 % et une température comprise entre 3 et 26°C pour que le développement et la libération des spores (sporulation) aient lieu. Sur des tissus hôtes sensibles, les spores peuvent germer directement, on parle alors de germination directe. La pénétration dans la plante permet la création de nouveaux sporanges à l'extérieur des tissus. Cette phase correspond à la sporulation. En fin de culture ou en cas de pluie, les sporanges peuvent tomber directement sur le sol en absence de surface foliaire disponible et sont entraînés par ruissellement d'eau jusqu'aux tubercules, assurant ainsi l'infection de ces derniers. Le parasite pénètre dans les tubercules et constitue une forme de conservation de l'agent pathogène pendant l'hiver et redémarre l'année suivante : soit par les repousses, avec des tubercules infectés laissés dans le sol pendant tout l'hiver ou dans les tas de déchets, les écarts de triage, soit par des tubercules infectés, sans symptôme visible, plantés l'année suivante. Il existe donc deux voies de contamination : par le sol et par les airs.

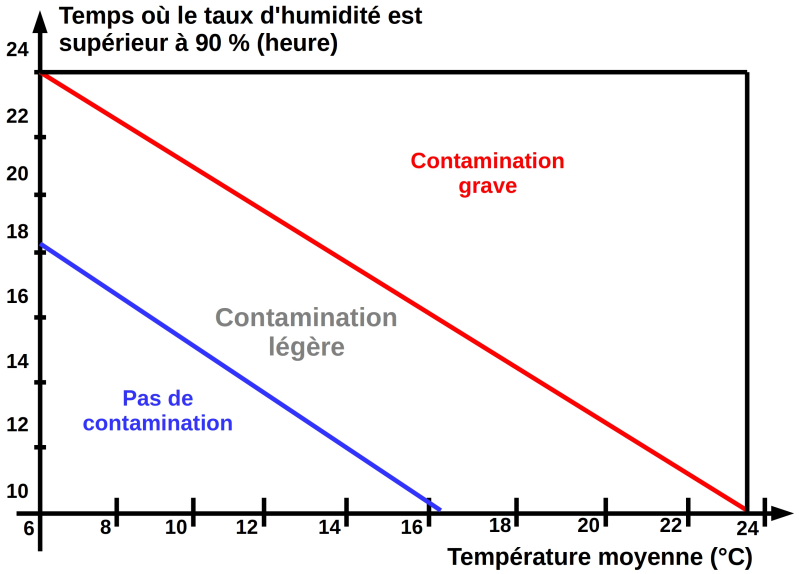


FIGURE 5.4 – Echelle de gravité des contaminations en fonction de la durée pendant laquelle l'humidité relative est supérieure à 90°% et de la température au sein du couvert végétal

Comme le montre très bien la figure 5.4, l'installation et l'évolution de la maladie sont très largement déterminées par les conditions climatiques, notamment l'humidité et la température. Dans les conditions optimales de développement de l'agent pathogène, la durée totale du cycle infectieux est de 3 à 5 jours. Les conditions environnementales nécessaires à l'installation de l'agent pathogène (entre 16 et 22°C et l'humidité relative supérieure à 90 %) sont souvent atteintes dans les grands bassins de production français et notamment en Bretagne et dans le Nord de la France.

5.1.3 Les moyens de traitement

La priorité de la lutte contre le mildiou est d'empêcher l'installation de la maladie et, le cas échéant, de réduire au maximum la vitesse de propagation de l'agent pathogène. Pour cela, différents moyens de traitement existent. Le premier consiste à effectuer un maximum de rotations culturales de façon à minimiser les sources d'agents pathogènes (inoculum primaire) et à éliminer les tas de déchets, issus de la récolte précédente, souvent laissés à proximité des parcelles. Ces tas de déchets constituent la source principale d'inoculum primaire (à savoir que dans des pays frontaliers comme la Belgique, il est obligatoire de bâcher les tas de pommes de terre pour limiter le risque d'infection). Les rotations culturales sont nécessaires pour éviter les infections par l'inoculum produit sur les repousses de tubercules laissées aux champs. Il reste cependant très difficile d'éliminer ces sources du fait de la difficulté à contrôler les repousses. La prévention des épidémies passe également par l'utilisation de tubercules de semences sains et la destruction des repousses.

Le principal moyen de traitement reste cependant l'utilisation de fongicides de contact, pénétrants ou systémiques. Différents fongicides sont utilisés aux cours du développement de la plante pour la protéger au mieux. À force d'utilisation, la souche du mildiou *P. infestant* est devenue résistante aux matières actives de certains fongicides par mutation génétique. Pour limiter cette résistance, il est conseillé d'utiliser certains fongicides uniquement comme produit de prévention. L'utilisation massive de pesticides et autres fongicides est à l'origine de la pollution des sols. C'est pourquoi, les gouvernements européens tentent d'en limiter l'usage.

La lutte préventive reste le meilleur moyen d'éviter la contamination d'une parcelle. C'est pourquoi, depuis le début des années 60, des moyens sont mis en place pour créer des outils d'aide à la décision reposant sur la simulation d'épidémie de mildiou à partir de données climatiques. Nos travaux s'inscrivent dans cette optique, de façon à pouvoir simuler des domaines spatiaux beaucoup plus grands.

5.2 Modélisation de la propagation du mildiou de la pomme de terre sur le territoire

La modélisation et la simulation de la propagation du mildiou de la pomme de terre sur le territoire nécessitent l'implémentation de plusieurs modèles. Il existe différents niveaux de complexité pour modéliser la prolifération et les effets du mildiou sur un ensemble de parcelles. Quelque soit le niveau de modélisation, le modèle global doit au minimum contenir un modèle de croissance de plante, offrant de préférence une réponse aux attaques éventuelles de mildiou, un modèle de développement des spores reprenant les étapes du cycle de génération du mildiou, présenté par la figure 5.3, et un modèle de dispersion des spores pour permettre sa prolifération. Ces modèles étant le cœur de la dynamique de propagation du mildiou de la pomme de terre, nous avons fait le choix de présenter, en détail, chacun d'entre eux. Cependant, pour espérer obtenir une modélisation de qualité, il est nécessaire de compléter le processus à l'aide d'autres modèles telles que la météo locale, les prévisions météorologiques, la gestion hydrique du sol ou la transpiration de

la plante. La gestion optimale de l'irrigation est aussi un facteur important de la réduction des risques. De même que l'utilisation d'outils de traitement géolocalisés est devenue indispensable. Il serait dommage de provoquer une infection suite à un mauvais passage des engins de traitement. L'ensemble de ces modèles seront brièvement présentés en section 5.2.4. Cette section montrera également notre apport à la modélisation du mildiou, notamment en ce qui concerne l'utilisation d'une météo localisée à proximité des lieux de culture (voire en bordure des parcelles).

5.2.1 Le modèle de dispersion gaussien *Plume*

Le modèle de Scherm [Sch96] est l'un des modèles de dispersion utilisé pour simuler la dispersion des spores de mildiou [Rak12]. En effet, il permet de déterminer le taux de dépôt des spores à une distance donnée d'une source. Ce modèle semble être limité par son traitement bi-dimensionnel de la dispersion. En effet, des modèles de dispersion tel que *Plume* [Hor77] offrent une vision tri-dimensionnelle beaucoup plus proche de la réalité du monde. C'est pourquoi, nous avons fait le choix d'utiliser ce dernier comme fonction de dispersion.

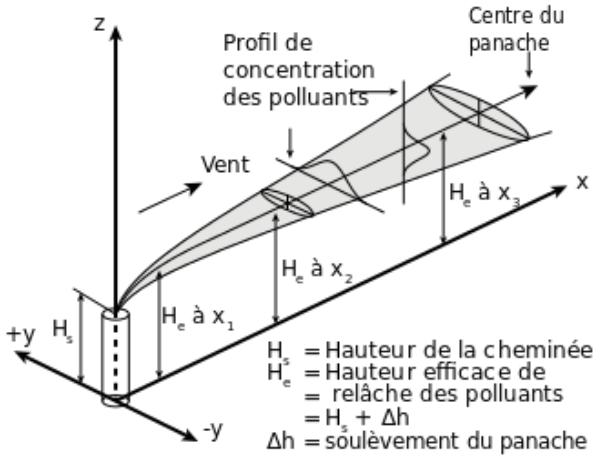


FIGURE 5.5 – Modèle gaussien de dispersion atmosphérique *"plume"* [source : *https ://fr.wikipedia.org*]

Le transport et la diffusion de spores dépendent du vent et de la turbulence atmosphérique d'origine mécanique ou thermique. En considérant la diffusion turbulente homogène et un champ de vent uniforme dans l'espace, la concentration de spores est supposée suivre une distribution gaussienne le long des plans perpendiculaires à la direction du rejet (voir la figure 5.5). Le modèle gaussien *Plume*, permet de déterminer la concentration de particules, en microgramme par mètre cube, en un point (x,y,z) à l'aide de l'équation suivante :

$$\chi(x,y,z,H) = \frac{Q}{2\pi\, U \sigma_y \sigma_z} \exp\left(-\frac{y^2}{2\sigma_y^2}\right)\left[\exp\left(-\frac{(z-H)^2}{2\sigma_z^2}\right) + \exp\left(-\frac{(z+H)^2}{2\sigma_z^2}\right)\right]$$

Où,

- Q est le taux d'émission de la source, en gramme par seconde
- U est la vitesse moyenne du vent à la source, en mètre par seconde

- σ_y et σ_z sont respectivement les écarts types des lois de distribution des concentrations horizontales et verticales, en mètre

- H est la hauteur de le source d'émission, en mètre

- x est la distance suivant l'axe x, en mètre

- y est la distance au centre de la gaussienne, en mètre

- z est la distance au sol, en mètre

Les paramètres σ_y et σ_z sont estimés en fonction de la stabilité du nuage et de la distance à la source grâce aux formules suivantes :

$$\sigma_y = cx^d \quad \text{et} \quad \sigma_z = ax^b$$

où, les paramètres a, b, c et d sont obtenus à partir des tableaux 5.1 et 5.2.

Classe de stabilité	**100 < x ≤ 500**		**500 < x ≤ 5000**		**5000 < x**	
	a	**b**	**a**	**b**	**a**	**b**
Très instable	0.0383	1.281	0.0002539	2.089	0.0002539	2.089
Instable	0.1393	0.9467	0.04936	1.114	0.04936	1.114
Légèrement instable	0.112	0.91	0.1014	0.926	0.1154	0.9109
Neutre	0.0856	0.8650	0.2591	0.6869	0.7368	0.5642
Légèrement stable	0.1094	0.7657	0.2452	0.6358	0.9204	0.4805
Stable	0.05645	0.8050	0.193	0.6072	1.505	0.3662

TABLEAU 5.1 – Coefficients permettant d'obtenir σ_z

Classe de stabilité	**x < 10000**		**x ≥ 10000**	
	c	**d**	**c**	**d**
Très instable	0.495	0.873	0.606	0.851
Instable	0.31	0.897	0.523	0.84
Légèrement instable	0.197	0.908	0.285	0.867
Neutre	0.122	0.916	0.193	0.865
Légèrement stable	0.0934	0.912	0.141	0.868
Stable	0.0625	0.911	0.08	0.884

TABLEAU 5.2 – Coefficients permettant d'obtenir σ_y

L'obtention de la quantité de spores au sol s'obtient lorsque $z = 0$. Dans ce cas, l'équation *Plume* devient :

$$\chi(x,y,H) = \frac{Q}{\pi U \sigma_y \sigma_z} \exp\left(-\frac{y^2}{2\sigma_y^2}\right) \exp\left(-\frac{H^2}{2\sigma_z^2}\right)$$

La figure 5.6, présente un exemple de dispersion de spores au sol et illustre par un panel de couleurs le taux de contamination dans cette zone.

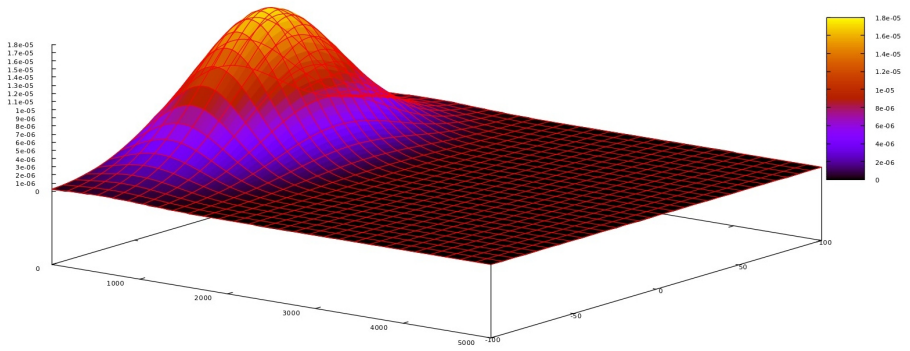


FIGURE 5.6 – Exemple de dispersion de spores au sol, à un instant t, en fonction de la distance à la source x, par utilisation d'un modèle gaussien *plume*

5.2.2 Le modèle de plante : Spudgro

Les modèles de culture sont des représentations mathématiques dont le rôle est de simuler le fonctionnement du système sol-plante-atmosphère sous l'influence de pratiques culturales. Ces relations mathématiques servent à estimer des grandeurs agronomiques, comme le rendement, la surface foliaire, etc ..., qui caractérisent ce fonctionnement en liaison avec des facteurs environnementaux physiques, chimiques, voire biologiques, et les techniques culturales. Ce fonctionnement du système sol-plante-atmosphère est composé et régit par de nombreux mécanismes qui ne peuvent être pris en compte dans leur ensemble. Par conséquent, les modèles de cultures reposent sur la hiérarchisation des processus biologiques, physiques et chimiques à représenter. Leurs structures dépendent des objectifs visés et des conditions de leur utilisation. De nombreux modèles de culture de pommes de terre existent, tels que LINTUL-POTATO [KH95], 2DSPUD [TKP+02] ou encore SPUDGRO [JJT86]. Ce dernier, développé par Johnson et al en 1986, est celui que nous avons choisi pour modéliser la croissance et le développement de la pomme de terre, à l'image du choix réalisé dans [Rak12]. En effet, le modèle Spudgro a l'avantage d'être un modèle simple de développement et de croissance dont le rôle est de simuler l'élaboration du rendement de la culture en fonction des conditions climatiques.

Acronyme	Unité	Description
		Paramètres
ERiL	-	Efficacité de répartition initiale pour les feuilles (= 0.08)
ERiR	-	Efficacité de répartition initiale pour les racines (= 0.16)
ERiS	-	Efficacité de répartition initiale pour la tige (= 0.16)
K2	$g.m^{-2}$	Paramètre de répartition de Michaelis-Menten pour LEAF (= 1.0)
K3	$g.m^{-2}$	Paramètre de répartition de Michaelis-Menten pour STEM (= 6.0)
K4	$g.m^{-2}$	Paramètre de répartition de Michaelis-Menten pour ROOT (= 1.0)
K6	1^{-1}	Taux d'utilisation journalier de la matière sèche accumulée ASSIM (= 0.75)
PNCGR	-	Taux d'accroissement potentiel de la plante
RLDM	-	Taux de remobilisation de la matière sèche des feuilles à la sénescence (= 0.50)

TABLEAU 5.3 – Acronymes et descriptions des paramètres de Spudgro

Acronyme	Unité	Description
		Variables d'entrée
D50	j	Date en jour julien où 50% des plants ont levé
K1	$g.m^{-2}$	Constante de répartition globale de Michaelis-Menten (= f(Var))
K5	$g.m^{-2}$	Constante de répartition de Michaelis-Menten pour TUBER (= f(Var))
SWP	bar	Potentiel hydrique du sol (Soil Water Potential)
Tmin	°C	Température maximale journalière
Tmax	°C	Température minimale journalière
TTmax	°C j	Age physiologique maximal de la plante
TTt	°C j	Age physiologique nécessaire pour débuter la tubérisation
Var	-	Variété
		Variables d'état
AP	Cj	Age physiologique (Francl, 1989)
ASSIM	$g.m^{-2}$	Biomasse sèche cumulée par unité de surface de sol
DMAc	-	Taux d'accroissement de matière sèche par unité de surface de sol et de temps
LAI	$m^2.m^{-2}$	Indice de surface foliaire (Leaf Area Index)
LEAF	$g.m^{-2}$	Biomasse sèche des feuilles par unité de surface de sol
Mil	-	Sévérité du mildiou (proportion de surface foliaire attaquée)
MSG	$g.m^{-2}$	Matière sèche initiale des plants par unité de surface de sol
MSRF	-	Facteur de réduction dû au stress hydrique (=f(SWP))
PTRi	-	Proportion de rayonnement total intercepté (= f(LAI))
ROOT	$g.m^{-2}$	Quantité de matière sèche des racines par unité de surface de sol
STEM	$g.m^{-2}$	Quantité de matière sèche des tiges par unité de surface de sol
TSR	-	Total Solar Radiation Rayonnement global journalier
TUBER	$g.m^{-2}$	Quantité de matière sèche des tubercules par unité de surface de sol

TABLEAU 5.4 – Acronymes et descriptions des variables de Spudgro

Le mécanisme du modèle

Le mécanisme de fonctionnement de ce modèle est présenté par le schéma 5.7. Nous pouvons distinguer quatre catégories de variables, classées par couleur :

- la marron, de forme irrégulière, correspond à la variable de source
- les vertes, de forme rectangulaire, correspondent aux variables d'état
- les bleus argentées, de forme ovale, correspondent aux variables intermédiaires
- les bleu ciel, de forme ovale, correspondent aux variables d'entrée

Ce mécanisme se compose également d'un taux d'accroissement de la biomasse, représenté en jaune sur le schéma, et nécessite la présence de paramètres, représentés par des ronds blancs sur le schéma. Nous pouvons également distinguer deux catégories de flèches :

- celles en trait plein représentent les flux de matière
- celles en traits pointillés représentent le flux d'information

La bonne compréhension de ce schéma, ainsi que de la suite de cette sous-section, nécessite l'utilisation des tableaux 5.3 et 5.4 qui référencent l'ensemble des acronymes et des définitions de variables et de paramètres.

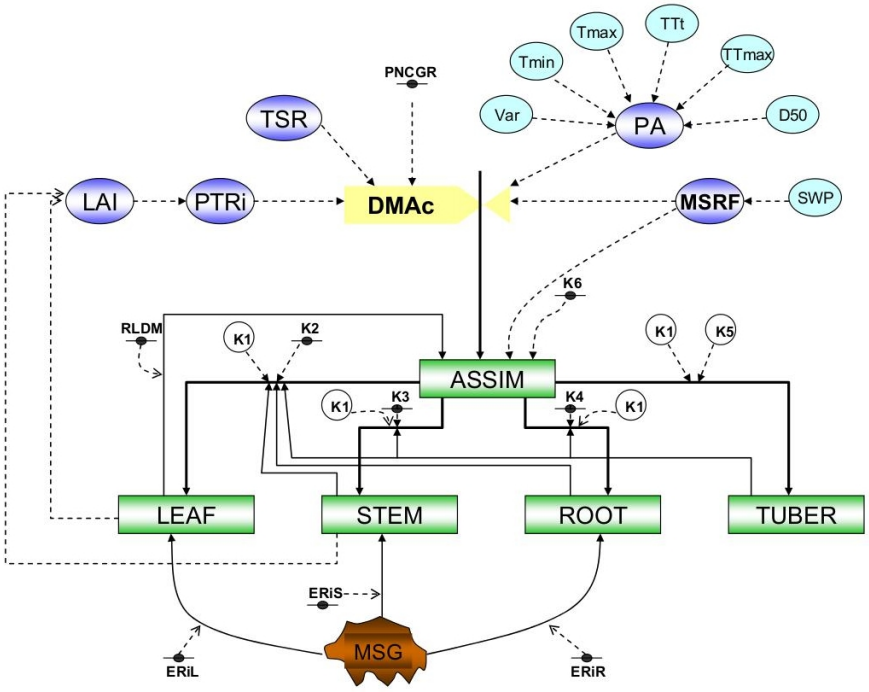


FIGURE 5.7 – Diagramme de Forrester de Spudgro [source : [Rak12]]

Le modèle Spudgro nécessite l'utilisation de variables d'entrée, représentées en bleu ciel sur le schéma 5.7, qui se composent de données météorologiques (température et radiation solaire), de données physiologiques caractérisant la croissance et le développement de la plante (durée de la croissance, durées des stades de développement végétatif et de tubérisation, précocité de la variété), et enfin une caractéristique physique du sol : l'évolution du potentiel hydrique (SWP : Soil Water Potential). A ce sujet, nous avons procédé à une légère modification afin de faire le lien avec un modèle de transpiration. Le modèle permet de calculer la quantité de matière sèche synthétisée ainsi que sa répartition dans la plante à un pas de temps journalier. Il permet également de calculer le rendement final à partir des conditions d'accumulation et d'évolution de la quantité de matière dans les différents organes de la plante, à la fin du développement de la culture.

Présentation des équations qui composent le modèle

Cette sous-section présente l'ensemble des équations contenues dans le modèle Spudgro, tout en respectant l'ordre établi par le schéma 5.7. L'accroissement de la matière sèche par unité de surface (DMAc) est obtenu à partir de l'équation suivante :

$$DMAc = PTRi * TSR * PNCGR * Cstr * AP$$

La proportion de rayonnement total intercepté (PTRi) dépend de l'indice de surface foliaire (LAI) qui est estimé à partir de la masse sèche des feuilles par unité de surface (LEAF) et des tiges (STEM). TSR (Total Solar Radiation) représente le rayonnement global journalier. PNCGR (Potential Net Crop Growth Rate) est le taux potentiel d'accroissement. Cstr (Moisture Stress

Reduction Factor) est un facteur de réduction dû au stress hydrique. Le stress hydrique est calculé à l'aide d'un modèle de transpiration (besoin en eau de la plante) et de la disponibilité en eau du sol. Ce modèle sera présenté plus tard.

Concernant la variable TSR, plusieurs voies sont possibles : soit on la considère comme une donnée provenant d'une mesure (il existe des capteurs pour la mesurer), soit on utilise un modèle. C'est cette dernière voie que nous avons utilisé. Nous n'allons pas décrire toutes les équations de ce modèle mais il repose sur le calcul de la trajectoire du soleil en fonction de la période dans l'année et de la durée d'ensoleillement. Les conditions climatiques imposent naturellement des conditions qui influencent la variable $DMAc$ via la variable AP. Cette variable est en général un cumul de températures, on parlera de degrés-jour. De nombreuses formulations existent. Certaines sont basées sur une simple somme à laquelle on retranche une base (6°C pour la pomme de terre, par exemple). Dans notre cas, nous avons utilisé une formulation un peu plus complexe qui permet de tenir compte à la fois d'un seuil bas et d'un seuil haut. Ces seuils permettent de bloquer la physiologie de la plante lorsque les températures sont en dehors.

$$\mathrm{AP} = \begin{cases} 10*(1-(T-21)^2/196) \text{ si } T >= 7 \text{ et } T < 21 \\ 10*(1-(T-21)^2/81) \text{ si } T >= 21 \text{ et } T < 30 \end{cases}$$

De plus, comme nous avons accès à des données toutes les 15 minutes via des capteurs, pour la variable AP, nous calculons une moyenne journalière. Il y a probablement mieux à faire. C'est l'une des pistes d'amélioration. La figure 5.8 illustre les effets des données sur la variable LAI.

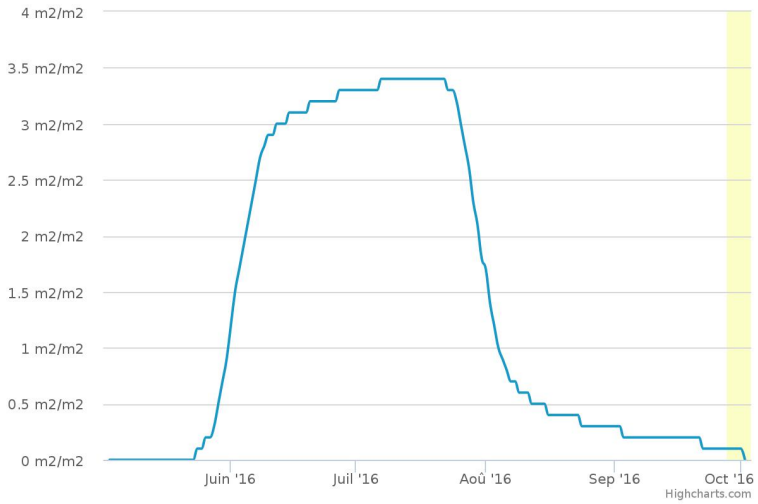


FIGURE 5.8 – Évolution de l'indice foliaire (LAI) avec les données météo de précision

Les variables $PTRi$ et LAI utilisées dans Spudgro sont définies à partir des équations ci-dessous.

$$\begin{aligned} PTRi &= \max(0, 1-e^{-Kdf*LAI}) \\ LAI &= LEAF(-1)*230+0.5*STEM(-1)*3.14 \end{aligned}$$

Dans ce cas, PTRi intègre l'équation d'interception lumineuse de Beer-Lambert [Lam60] [Bee52].

La quantité de matière calculée sur une plante moyenne est accumulée dans le réservoir ASSIM avant d'être répartie dans les différents organes de la plante tels que les feuilles (LEAF), la tige (STEM), les racines (ROOT) et le tubercule (TUBER). L'accroissement de la matière sèche par unité de surface fabriquée par jour vers les différents organes de la plante est calculé par les équations modifiées de Michaelis-Menten :

$$\begin{aligned} dL &= K2*(K1/(K1+STEM+ROOT+TUBER)) \\ dS &= K3*(STEM/(STEM+K1*(1+TUBER))) \\ dR &= K4*(ROOT/(ROOT+K1*(1+TUBER))) \\ dT &= K5*(TUBER/(TUBER+K1)) \end{aligned}$$

où dL représente l'accroissement journalier de biomasses par unité de surface dans les feuilles, dS représente l'accroissement journalier de biomasses par unité de surface dans la tige, dR représente l'accroissement journalier de biomasses par unité de surface dans les racines, et dT représente l'accroissement journalier de biomasse par unité de surface dans le tubercule.

La quantité de matière effectivement reçue par un organe à un instant donné est la proportion de quantité de matière prévue pour cet organe par rapport à la quantité totale de matière répartie pour toute la plante multipliée par la quantité totale de matière accumulée dans ASSIM à cet instant. L'accroissement de la quantité de matière est donné par :

$$dLEAF/dt = (dL/(dL+dS+dR+dT))*K6*ASSIM$$

L'évolution de cette quantité dans le temps est l'addition des matières déjà cumulées au niveau de l'organe par celles nouvellement apportées :

$$LEAF(t) = \Delta LEAF + LEAF(t-1)$$

Le modèle prend également en compte le fait que la quantité de matière sèche contenue dans les semences de pommes de terre constitue la quantité initiale (MSG) de matière allouée aux ébauches des différents organes de la plante.

5.2.3 Le modèle de développement des spores : Milsol

Les modèles épidémiologiques concernant le mildiou de la pomme de terre sont développés depuis 1950. On peut citer par exemple le modèle BLITECASTE [RWF93] et LATEBLIGHT [APHF+05]. Ils ont généralement pour objectif d'optimiser le nombre et les dates d'interventions pour les applications de fongicides en simulant l'évolution de la maladie. Cependant, le choix du modèle épidémiologique s'est porté sur ceux développés et utilisés, de longue date par le Service de la Protection des Végétaux (Ministère de l'agriculture) et plus récemment Arvalis. C'est notamment le cas du modèle Milsol, proposé par G. Lechapt en 1985 [Lad88]. Il simule, avec un pas de temps horaire, le niveau de risque de mildiou en se basant sur le calcul du nombre de spores vivantes présentes sur le feuillage de la culture et permet de cette manière une quantification de

l'épidémie. À l'image du schéma 5.3, Milsol comporte 4 compartiments selon le cycle épidémique du mildiou P. infestans.

Survie des spores et contamination

La contamination est représentée par une variable notée UDESPO (unité de développement des spores, grandeur adimensionnée) qui correspond au développement accompli par les spores en 1 heure à 21°C (conditions optimales). L'accroissement horaire de la contamination correspond à l'augmentation des unités de développement des spores (UDESPO) par unité de temps et s'écrit :

$$\begin{aligned} TUDESPO(\theta) &= p_1\theta + p_2 \text{ si } \theta \leq 18C \\ TUDESPO(\theta) &= p_3 \text{ si } \theta > 18C \end{aligned}$$

Les différents acronymes utilisés dans cette sous-section sont présentés et définis dans le tableau 5.5.

Acronyme	Unité	Description
		Paramètres
D0	-	Degré de développement pour le début de la germination (=100)
D1	-	Degré de développement pour le taux de contamination totale (=150)
Dc	-	Degré de développement pour le taux de survie nulle (=100)
p_1	$h^{-1}C^{-1}$	Coefficient d'unités de développement de spores $\theta \leq 18$ °C (=0.5)
p_2	h^{-1}	Coefficient d'unités de développement de spores $\theta \leq 18$ °C (=1.0)
p_3	-	Coefficient d'unités de développement de spores $\theta > 18$ °C (=10)
		Variables d'entrées
Hr	%	Humidité relative
θ	°C	Température moyenne horaire
		Variables d'état
CUMDDS	-	Degré de développement des spores
GRAVI	-	Proportion de spores ayant germées
POIDS	-	Nombre de spores contaminatrices
SPORES	-	Nombre de spores issues de la contamination primaire
SURVIE	-	Taux de survie des spores
TUDESPO	h^{-1}	Taux d'accroissement des unités de développement

TABLEAU 5.5 – Acronymes et descriptions des composants de Milsol, pour la survie des spores et leur contamination

Le degré de développement des spores (CUMDDS) se calcule à l'aide de la formule suivante :

$$CUMDDS(n) = \sum_{h=1}^{n} TUDESPO(\theta(h))$$

où, n est le nombre d'heures écoulées depuis l'instant initial.
Le taux de contamination (GRAVI) dépend des conditions d'hygrométrie et du degré de dévelop-

pement des spores (CUMDDS(n)). Il s'obtient à partir des formules suivantes :

$$\begin{aligned} GRAVI(n) &= (CUMDDS(n) - D0)/(D1 - D0) \text{ si } Hr > 90\% \\ GRAVI(n) &= 0 \text{ si } Hr < 90\% \end{aligned}$$

Le nombre de spores contaminatrices (POIDS) est calculé à l'aide de la formule suivante :

$$POIDS = GRAVI * SPORES$$

Le taux de survie horaire (SURVIE) dépend des conditions d'hygrométrie et du degré de développement de spores CUMDDS. Quand les conditions sont favorables, toutes les spores survivent et une partie d'entre elles vont germer si les conditions de contamination sont réunies.

$$\begin{aligned} SURVIE &= 1 \text{ si } Hr > 90\% \\ SURVIE &= 1 - CUMDDS/Dc \text{ si } Hr < 90\% \text{ et } CUMDDS \leq Dc \\ SURVIE &= 0 \text{ si } Hr < 90\% \text{ et } CUMDDS > Dc \end{aligned}$$

Incubation et sporulation potentielle

Le développement du mycélium (INCUB) dans les tissus en 1 heure à 21 °C dépend uniquement de la température. L'augmentation des unités de développement du mycélium par unité de temps s'écrit :

$$\begin{aligned} TINCUB(\theta) &= p_4.\theta \text{ si } \theta \leq 18C \\ TINCUB(\theta) &= p_4.\theta - RET(\theta) \text{ si } \theta > 18C \end{aligned}$$

Jusqu'à 18 °C, le taux d'incubation croît linéairement. Cependant, les températures supérieures à 18 °C sont considérées comme préjudiciables à la croissance du mycélium. Au-delà de cette valeur, INCUB subit une décroissance correspondant au retard de croissance RET :

$$RET(\theta) = p_5(\theta - \theta_{opt})^{p_6}$$

Le degré de développement du mycélium dans les tissus est calculé par :

$$AGE(n) = \sum_{h=1}^{n} TINCUB(\theta(h))$$

Où n est le nombre de demi-journées écoulées depuis l'instant initial.
La capacité de sporulation (KASPO) est le potentiel de sporulation d'une lésion occasionnée par une spore en fonction de l'AGE. Elle est donc dépendante de la variable AGE.

$$\begin{aligned} KASPOthéorique &= FACT(AGE - p_7)/p_7 \text{ si } p_7 < AGE \leq p_8 \\ KASPOthéorique &= FACT(p_9 - AGE)/p_9 \text{ si } p_8 < AGE < p_9 \end{aligned}$$

Les différents acronymes utilisés dans cette sous-section sont présentés et définis dans le tableau 5.6.

Acronyme	Unité	Description
		Paramètres
FACT	-	Maximum du potentiel de sporulation
p_4	$h^{-1}C^{-1}$	Coefficient d'unité d'incubation (=0.05)
p_5	$h^{-1}C^{-1.5}$	Coefficient d'unité d'incubation (=0.025)
p_6	-	Coefficient d'unité d'incubation (=1.5)
p_7	-	Coefficient d'unité d'incubation (=75)
p_8	-	Coefficient d'unité d'incubation (=150)
p_9	-	Coefficient d'unité d'incubation (=225)
p_{10}	$h^{-1}C^{-1.5}$	Coefficient du retard sur le développement (=0.004)
θ_{opt}	°C	Température optimale pour l'incubation (=18)
		Variable d'entrée
θ	°C	Température moyenne d'une demi-journée
		Variables d'état
AFFAIB	-	Degré de réduction du potentiel de sporulation
AGE	-	Degré de développement du mycélium dans les tissus
KASPOréel	-	Potentiel de sporulation réel
KASPOthéo	-	Potentiel de sporulation théorique
POSPO	-	Sporulation potentielle des lésions
RET	h^{-1}	Retard sur la croissance
SPOSPO	-	Sporulation potentielle de tous les cycles
TINCUB	h^{-1}	Taux d'accroissement des unités d'incubation
TRED	h^{-1}	Taux de réduction du potentiel de sporulation

TABLEAU 5.6 – Acronymes et descriptions des composants de Milsol, pour l'incubation et sporulation potentielle

Les températures supérieures à 18 °C ont un effet réducteur sur le potentiel de sporulation (RED). Cette réduction de temps s'écrit :

$$TRED(\theta) = FACT * p_{10}(\theta - \theta_{opt})^{p_6}$$

Le degré de réduction du potentiel de sporulation (AFFAIB) est calculé par la formule suivante :

$$AFFAIB(n) = \sum_{h=1}^{n} TRED(\theta(h))$$

Où n est le nombre de demi-journées écoulées depuis l'instant initial. Le potentiel de sporulation réel s'écrit :

$$KASPOréelle = KASPOthéorique - AFFAIB$$

La sporulation potentielle des lésions occasionnées par l'ensemble des spores contaminatrices (POSPO) est calculée par période de 12 heures comme suit :

$$POSPO = KASPOréelle * POIDS$$

Le potentiel de sporulation (SPOSPO) est la somme des sporulations potentielles de tous les cycles effectués :

$$SPOSPO = \sum_{h=1}^{n} POSPO$$

Sporulation réelle

La sporulation est représentée par une variable USPORU. Elle correspond au développement de la sporulation accompli en 1 heure à 21 °C. Le taux d'accroissement des USPORU s'écrit :

$$TUSPORU(\theta) = p_{11}(\theta - \theta_{min})^{p_{12}} [p_{13} - p_{14}(\theta - \theta_{min})]$$

Les différents acronymes utilisés dans cette sous-section sont présentés et définis dans le tableau 5.7. Le nombre d'unités de développement de la sporulation (CUMSPO) est calculé par la

Acronyme	Unité	Description
		Paramètres
p_{11}	$°C^{-2}$	Coefficient d'unités de développement de la sporulation (=0.009)
p_{12}	-	Coefficient d'unités de développement de la sporulation (=2)
p_{13}	h^{-1}	Coefficient d'unités de développement de la sporulation (=1)
p_{14}	$h^{-1}\ C^{-1}$	Coefficient d'unités de développement de la sporulation (=0.037)
θ_{min}	°C	Température minimum pour la sporulation (=3)
		Variables d'entrée
Hr	%	Humidité relative
θ	°C	Température moyenne horaire
		Variables d'état
ACTISPO	-	Activité de sporulation
CUMSPO	-	Nombre d'unités de développement de la sporulation
SPORUL	-	Nombre de spores prêtes à être disséminées
TUSPORU	h^{-1}	Taux d'accroissement des unités de développement de la sporulation

TABLEAU 5.7 – Acronymes et descriptions des composants de Milsol, pour la sporulation réelle

formule suivante :

$$CUMSPO(n) = \sum_{h=1}^{n} TUSPORU(\theta(h))$$

Où n est le nombre d'heures écoulées depuis l'instant initial. La variable ACTISPO est la proportion de spores effectivement produites par rapport à la sporulation potentielle. Elle est calculée par la formule :

$$ACTISPO = \frac{CUMSPO - CUM0}{CUM1 - CUM0}$$

Où CUM 0 est un paramètre adimensionné correspondant à la valeur de CUMSPO à partir de laquelle débute l'activité de sporulation et où CUM 1 correspond à la valeur de CUMSPO à partir de laquelle l'activité de sporulation atteint son maximum.
La sporulation réelle est représentée par la variable SPORUL. Elle correspond au nombre de spores

produites et prêtes à être dispersées.

$$SPORUL = ACTISPO * SPOSPO$$

La dernière partie du modèle Milsol s'avère être la fonction de dispersion des spores. Dans notre cas, nous avons fait le choix d'utiliser une méthode de dispersion gaussienne *Plume*, présentée en section 5.2.1.

Le mécanisme du modèle

Le mécanisme de fonctionnement de ce modèle est présenté par le schéma 5.9. À l'image du modèle Spudgro, nous pouvons distinguer trois catégories de variables, classées par couleur :

- les vertes, de forme rectangulaire, correspondent aux variables d'état
- les bleus argentées, de forme ovale, correspondent aux variables intermédiaires
- les bleus ciel, de forme ovale, correspondent aux variables d'entrée

Ce mécanisme se compose également d'un taux d'accroissement de la biomasse, représenté en jaune sur le schéma, et nécessite la présence de paramètres, représentés par des ronds noirs. Nous pouvons également distinguer des flèches en pointillés représentant les flux d'information.

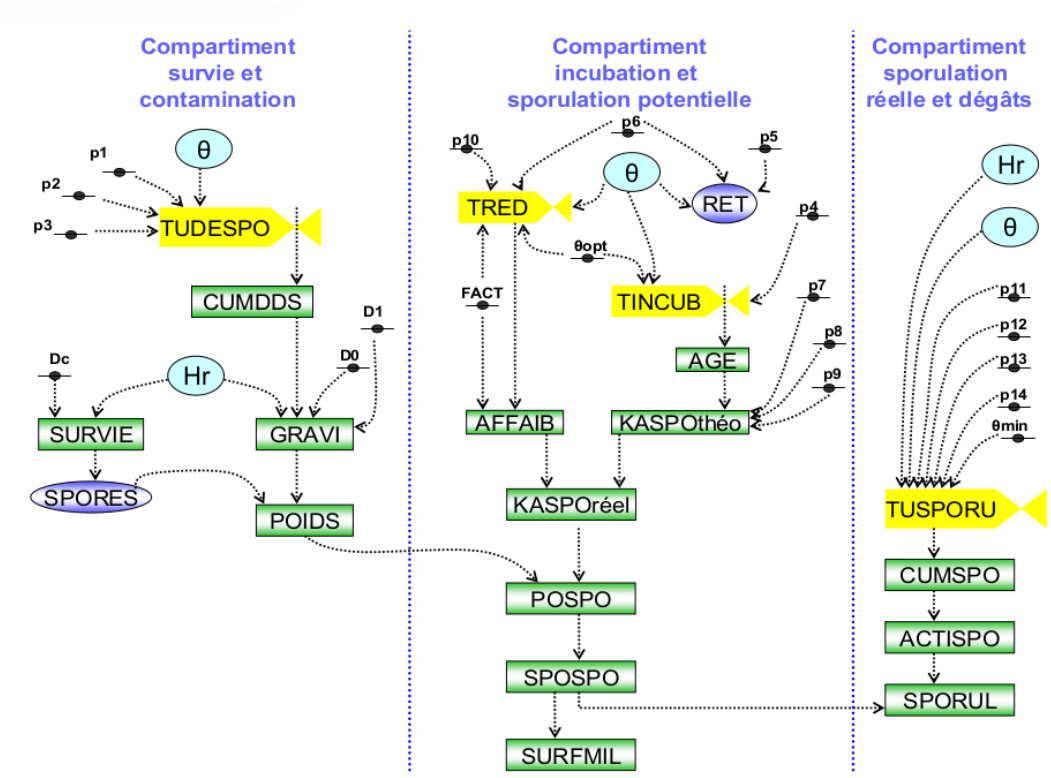


FIGURE 5.9 – Diagramme de Forrester de Milsol [source : [Rak12]]

5.2.4 Les modèles complémentaires

Les modèles Spudgro et Milsol sont tout deux dépendants, ou influencés par des éléments extérieurs telles que la météo ou les informations liées au sol. Il est donc intéressant de se pencher sur la création de modèles permettant de fournir, avec une très grande fiabilité, ces informations. En effet, plus les données fournies aux modèles Spudgro et Milsol sont de qualité, plus celles transmises par ces derniers seront de qualité. Le rôle de cette section est de présenter (sans entrer dans le détail), d'une part, nos apports en ce qui concerne le modèle météorologique et, d'autre part, de présenter les modèles de gestion du sol et du stress hydrique et leurs interactions avec le modèle Spudgro.

Le modèle météo de précision

La connaissance de la météo est fondamentale, tant pour le développement de la plante que pour la dispersion du mildiou. D'où l'importance de posséder une météo fiable et de précision. La météo, au sens large, regroupe l'hydrométrie, la pluviométrie, la température, la direction et la puissance du vent et peut également fournir le rayonnement solaire. Jusqu'à aujourd'hui, la météo nécessaire au fonctionnement des modèles Spudgro et Milsol était récoltée à partir de la station météorologique la plus proche, pouvant se trouver à des dizaines de kilomètres des cultures. Or, même si le climat reste globalement le même sur une courte distance, il est tout à fait possible d'observer des changements au niveau du vent et de la température. En effet, comme le montre l'exemple de la figure 5.10, les cultures sont réparties sur différents sites du territoire : une sur la côte au niveau de Oye-Plage, une dans les terres au niveau de Les Attaques et d'autres dans les environs (Éperlecques, Loon-Plage, etc ...).

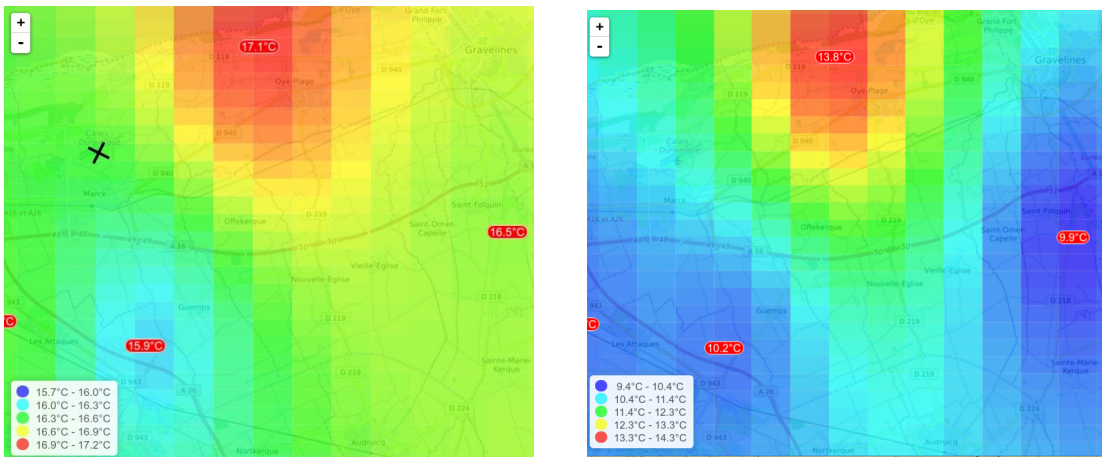


FIGURE 5.10 – Évolution des températures sur le territoire en fonction de la localisation des cultures. Température à 14h et à 23h le 25/09/2016

Chaque culture possède sa propre station météo (capteur de température, d'hydrométrie, de pluviométrie et de puissance de vent). Une station météo (de Météo France) est postée au niveau de l'aéroport de Calais-Dunkerque dans la ville de Marck. Cette station se situe à une dizaine de kilomètres des cultures et est représentée par une croix noire sur la figure de gauche. Dans ces

exemples, nous pouvons constater que les températures relevées sur les différents sites ne sont pas équivalentes. En effet, il existe un écart d'environ 1.2°C entre les températures relevées sur le littoral et dans les terres à 14h. Ce phénomène semble être amplifié la nuit (23h). Dans ce cas, l'écart maximum observé est de 3.9°C. Cet exemple, montre que, même sur une courte distance, la température peut considérablement varier ce qui justifie l'importance d'une météo de précision pour garantir une simulation fiable. Il en va de même pour les autres facteurs liés à la météo.

Pour ne plus dépendre d'une météo localisée au niveau du territoire (un cercle d'une cinquantaine de kilomètres), certaines entreprises à l'image de Weenat, offrent toute une gamme de capteurs géolocalisés permettant de fournir une information fiable en temps réel (une mesure toutes les 15 minutes). En effet, l'utilisation d'objets connectés permet de récolter plus facilement et rapidement des données. Pour garantir la fiabilité de celles-ci et afin d'éviter une éventuelle dérive de capteur, chacun d'entre eux est déployé plusieurs fois sur un même site. Afin de garantir une précision inégalable, ces capteurs sont installés à même le champ, ou très proches de celui-ci lorsqu'il s'agit d'un pluviomètre. Au delà de la météo, ces capteurs sont également capables de fournir la quantité d'eau contenue dans le sol (réserve utile) par le biais d'un tensiomètre. Il existe différentes tailles de capteurs (10 et 30 cm) de façon à pouvoir capter le taux d'humidité à différentes profondeurs. La pomme de terre utilisant l'eau en surface (moins de 30 cm), il est nécessaire d'utiliser des tensiomètres de 10 cm. La figure 5.11 présente un ensemble de capteurs fournis par la société Weenat. Parmi ces capteurs, nous pouvons trouver un tensiomètre, un hydromètre, un thermomètre et un pluviomètre. D'autres capteurs existent à l'image de l'anémomètre (vent) ou du pyranomètre (rayon lumineux).

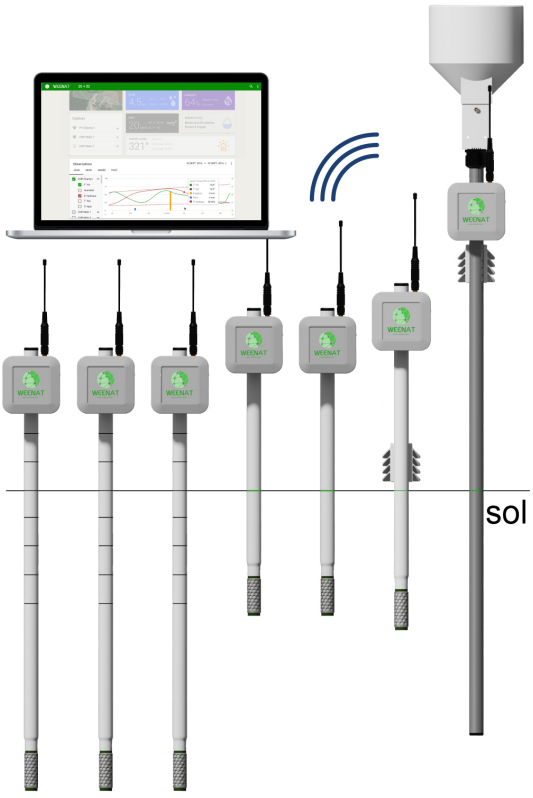


FIGURE 5.11 – Ensemble de capteurs fournis par la société Weenat [sources : *http ://www.weenat.com*]

En plus de fournir une donnée plus précise, l'aspect temps réel de ces capteurs permet de réaliser un re-calibrage des modèles à chaque relevé. Ainsi, les modèles Spudgro et Milsol possèdent un meilleur paramétrage et seront par conséquent susceptibles de fournir un meilleur résultat.

Le modèle de stress hydrique

Le modèle Spudgro (modèle de plante) est à la fois soumis à la météorologie (température, principalement) et à la disponibilité en eau dans le sol. En effet, la version originale du modèle contient une variable externe nommée SWP. Cependant, nous l'avons remplacé par la variable Cstr qui est beaucoup plus précise et qui correspond au coefficient de réduction de l'assimilation en fonction du déficit hydrique. Cette variable dépend à la fois de l'état de la réserve utile pour le niveau d'horizon du système racinaire (SWC et $FTSW$, le pourcentage de remplissage de la réserve utile) et de la transpiration potentielle de la plante ($TransPot$). La réserve utile est calculée dans le modèle de sol, présenté en section 5.2.4. La transpiration potentielle est fonction du coefficient cultural et indirectement de l'interception lumineuse. Cette dernière dépend directement de l'indice foliaire fourni par le modèle de plante.

Au cœur du calcul de la transpiration potentielle, on retrouve une estimation de l'évapotranspiration Eto (la perte d'eau liée aux conditions climatiques). Cette dernière est basée sur les équations de Penman Monteith avec la méthode définie par la FAO (Organisation des Nations Unies pour l'alimentation et l'agriculture). La FAO fournit aussi les valeurs des coefficients culturaux (kc_{min}, kc_{max} et kc_{pot}) pour la plupart des grandes espèces de plante, dont la pomme de terre [Pat] [Rad].

$$
\begin{aligned}
Interc &= 1 - e^{Kdf \times lai} \\
FTSW &= SWC / SWC_{max} \\
TransPot &= Kc \times Eto \\
pFact &= \min(0.8, \max(0, pFactor + 0.04(5 - TransPot))) \\
Cstr &= \max(0, \min(FTSW/(1 - pFact), 1))
\end{aligned}
$$

Où Kc s'obtient comme suit :

$$
Kc = \begin{cases} Kcmin \text{ si } Interc = 0 \\ \max(Kcmin, KcPot \times Interc) \text{ si } Ftsw > 0.75 \\ KcPot \times Interc \text{ sinon} \end{cases}
$$

Où, $pFact$ est un facteur de réduction lié à l'évapotranspiration dépendant d'une constante $pFactor$.

Il est alors possible de calculer une estimation de l'évapotranspiration réelle par la formule suivante :

$$
\text{Transpiration} = \begin{cases} Kc \times Eto \text{ si pluie} < 0.5 \\ 0 \text{ sinon} \end{cases}
$$

Où Kc s'obtient comme suit :

$$\text{Si } Interc > 0$$

$$Kc = \begin{cases} \max(Kcmin, KcPot \times Interc \times Cstr) \text{ si } Ftsw > 0.75 \\ KcPot \times Interc \times Cstr \text{ sinon} \end{cases}$$

$$\text{Sinon}$$

$$Kc = Kcmin$$

Où, $Kcmin$ est le coefficient cultural minimal , $KcPot$ est le coefficient cultural potentiel, $Cstr$ est le coefficient de stress hydrique, $FRWS$ est le pourcentage de remplissage de la réserve utile et Eto est l'évapotranspiration de base.

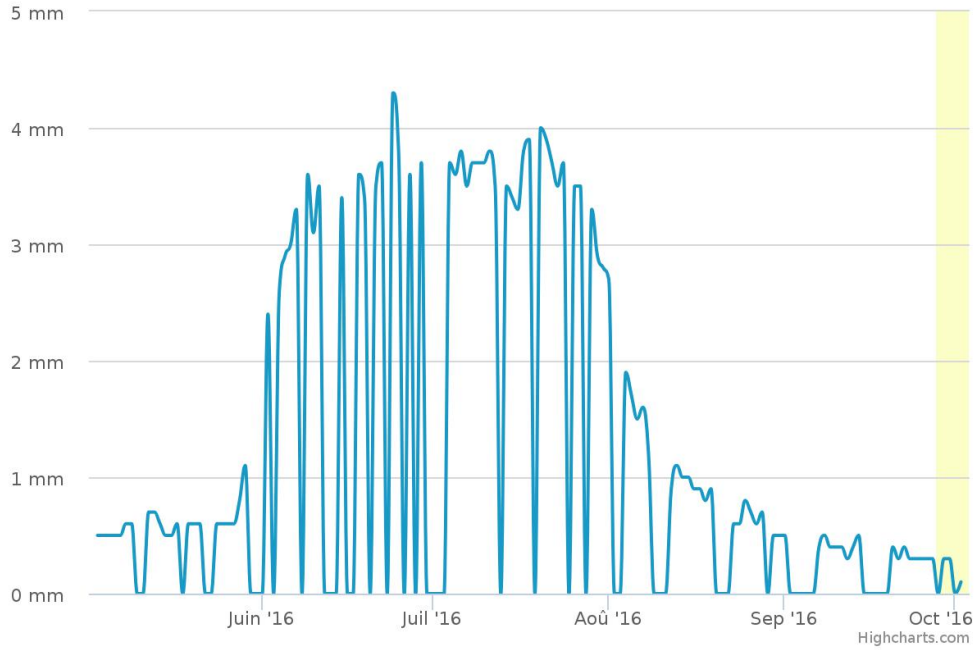


FIGURE 5.12 – Évolution de l'évapotranspiration

La figure 5.12 montre l'évolution de l'évapotranspiration sur une parcelle de pomme de terre localisée aux Attaques près de Calais. Cette évapotransporation utilise les mesures d'une station météo placée à côté de la parcelle.

Le modèle de sol

Le calcul de la réserve utile est intimement lié à un modèle de sol. Dans une première approximation, nous avons assimilé le sol à un unique réservoir. Mais, nous nous sommes aperçus assez rapidement que ce modèle avait atteint ses limites. C'est pour cela que nous nous sommes orientés vers un modèle en couches. Le sol est donc discrétisé en couche de 10 cm. Les processus de ruissellement, d'infiltration et de remontée capillaire sont modélisés. De plus, suite à des observations terrains, nous avons ajouté un processus de constitution de "flaques" d'eau car la période mai-juin 2016 a fait l'objet de fortes pluies ce qui a conduit à l'apparition de ce phénomène.

Les variables de sortie de ce modèle permettent d'estimer l'état de la réserve utile dans chacune des couches.

$$SWC = \sum_{z<\text{rootdepth}} Vol(z) - Wrd(z)$$
$$SWC_{max} = \sum_{z<\text{rootdepth}} Vol(z)$$

Où, *rootdepth* est la profondeur racinaire, *SWC* (Soil Water Content) correspond à la réserve utile, SWC_{max} est la réserve utile maximum, $Vol(z)$ correspond au volume de la couche de sol à la profondeur z et $Wrd(z)$ (Water reserve deficit) correspond la quantité d'eau manquante dans la réserve utile de la couche de profondeur z.

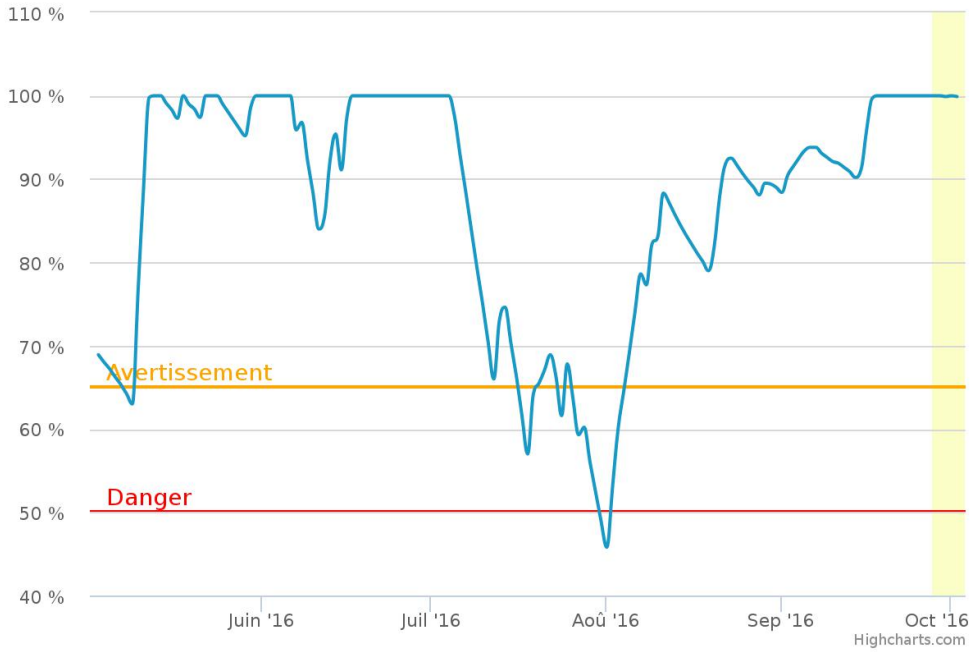


FIGURE 5.13 – Évolution de la réserve utile (SWC)

Ce modèle de sol est en cours de validation à l'aide de capteurs de type tensiomètre. En effet, les mesures issues des capteurs permettent de comparer les sorties du modèle de sol et de faire une estimation de la réserve utile à l'aide de 3 tensiomètres placés à 3 profondeurs différentes. Par intégration des mesures sur l'horizon racinaire, il est alors possible de faire une estimation assez précise de la réserve utile. Nous avons alors montré que le modèle était plus réactif aux apports d'eau que les capteurs et qu'un écart en valeur absolue est observé. Nous allons donc passer à une autre phase : la recalibration du modèle à l'aide des mesures. Il sera alors possible d'ajuster les paramètres du sol et de la plante afin de minimiser ces écarts. De plus, les agriculteurs partenaires du projet ont identifié sur le terrain les différents stades de la plante, ce qui nous offre une autre mesure pour encore mieux ajuster les modèles. Tous ces efforts ont un unique objectif : améliorer la précision des modèles.

5.3 Simulation de la dynamique du mildiou sur un territoire

Dans les sections précédentes, chaque sous-modèle a été présenté de manière indépendante. Nous allons maintenant nous intéresser aux interactions et donc au couplage de ces différents sous-modèles dans un contexte de simulation à l'échelle d'un territoire. Le territoire est représenté sous forme d'un ensemble de parcelles agricoles et de "vides". Tout élément qui n'est pas une parcelle agricole est ignoré : route, étendue d'eau, bâtiment, ... La figure 5.14 représente le parcellaire de la commune de Oye-Plage soit 404 parcelles pour une surface totale de 2243 hectares. Naturellement, seule une partie est consacrée à la culture de la pomme de terre. Néanmoins, pour nos simulations, nous avons considéré que toutes les parcelles sont dédiées à la culture de la pomme de terre. Dans le cas de la zone étudiée, seuls 150 hectares sont occupés par des cultures de pommes de terre. Chaque parcelle est subdivisée en cellule de 20m $\times$ 20m, ce qui conduit à travailler sur 58128 cellules pour le parcellaire complet.

FIGURE 5.14 – Parcellaire de la commune de Oye-Plage (représenté en couleur sombre)

5.3.1 Composition des modèles atomiques

Le graphe de modèles se décompose en trois niveaux (voir figure 5.15 et figure 5.16) :

— les connexions inter-parcelles : les échanges entre les parcelles se limitent aux données liées à la quantité de spores actives dans l'ensemble des cellules d'une parcelle. Ces quantités sont transmises aux parcelles présentes dans une zone décrite par un cercle dont le centre est le barycentre de la parcelle et dont le rayon est égal à la distance maximale de dispersion des spores. Chacune des cellules de la parcelle calcule la quantité de spores en provenance de chacune des parcelles voisines, à l'aide de la fonction de dispersion. Ce niveau est illustré par la figure 5.15.

— les connexions intra-parcelles (entre les cellules) : les échanges de spores ne se limitent pas aux échanges inter-parcelles. Une cellule peut être infectée par des cellules de la même

parcelle. Ceci permet de modéliser la progression du mildiou au sein même d'une parcelle. La qualité de la modélisation dépend bien évidemment de la discrétisation spatiale.

— les connexions entre les modèles atomiques : les modèles atomiques sont divisés en deux sous-ensembles. Le premier sous-ensemble correspond aux processus au niveau de chaque cellule (dynamique du mildiou et dispersion) et le second au niveau de la parcelle (dynamique de la plante et du sol). En effet, les processus liés à la plante et au sol sont uniques pour l'ensemble de la parcelle. La figure 5.16 illustre ces modèles atomiques et leurs connexions. On part de l'hypothèse forte que la parcelle est homogène en terme de climat et de sol. Ce qui naturellement est probablement faux, au moins sur le sol.

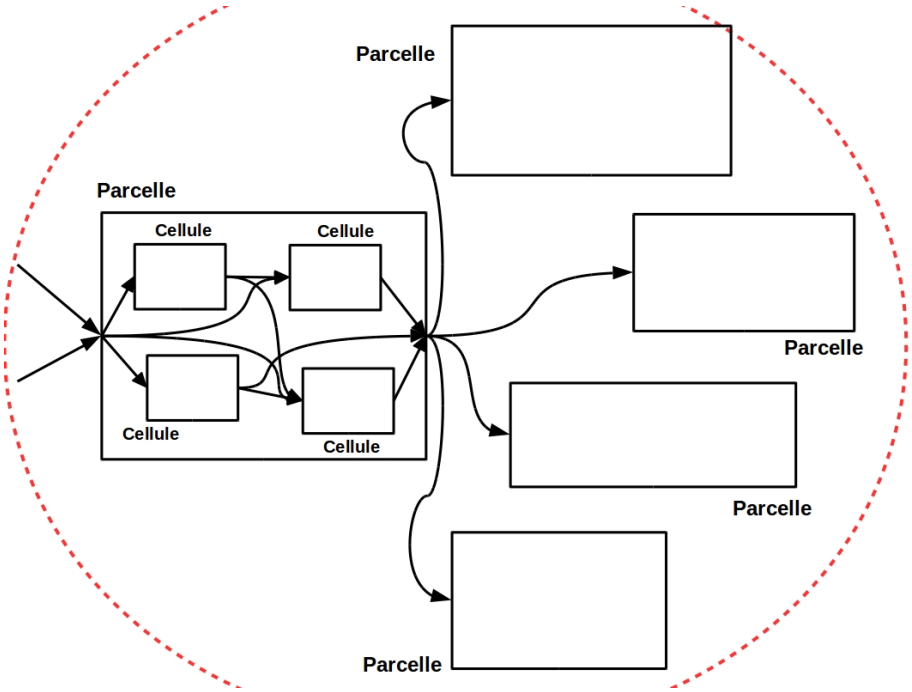


FIGURE 5.15 – Graphe de connexions inter et intra-parcelles

Détaillons un peu plus en détail le graphe de modèles de la figure 5.16. Comme nous l'avons développé dans la première partie, il est important de prendre en compte l'ensemble des processus mis en jeu dans le développement du mildiou. Pour cela, on retrouve sur la figure 5.16 l'ensemble des modèles en charge des différents processus (plante, sol et maladie). Le point d'entrée est constitué par les conditions climatiques via le modèle Météo. Ce modèle est issu des données acquises par mesure, à l'aide des capteurs placés au niveau de la parcelle. Dans le meilleur des cas, chaque parcelle est équipée d'une station météorologique donnant accès à la température de l'air à 2m, l'hydrométrie et la pluviométrie. Naturellement, pour des aspects financiers, le nombre de stations est limité et les données d'une station peuvent être utilisées pour plusieurs parcelles considérées comme très proches (quelques centaines de mètres). Le deuxième point d'entrée est le modèle de rayonnement solaire (RG). Dans le cadre de cette étude, le rayonnement solaire est calculé à partir de la durée moyenne d'ensoleillement, ainsi que le barycentre de la parcelle.

A partir de ces 2 modèles climatiques, une estimation de l'évapotranspiration potentielle est calculée à l'aide du modèle Eto. Ce modèle admet une entrée supplémentaire : la vitesse du vent.

Cette information complétée par la direction est aussi une entrée du modèle de dispersion. Au jour d'aujourd'hui, les données de vent sont relativement imprécises. En effet, nous utilisons les données mesurées à l'aéroport de Calais-Dunkerque, à Marck qui se situe juste à l'ouest de la zone étudiée (voir figure de gauche 5.10). Pour la prochaine campagne, il sera possible d'utiliser des données plus précises par l'intermédiaire d'anémomètres développés par la société Weenat. Il faudra alors choisir avec intelligence la position de ces anémomètres sachant que la zone d'études contient des sous-zones très différentes, pouvant faire l'objet de turbulences faussant ainsi les résultats. C'est notamment le cas pour certaines parcelles se trouvant derrière des obstacles, comme des forêts par exemple.

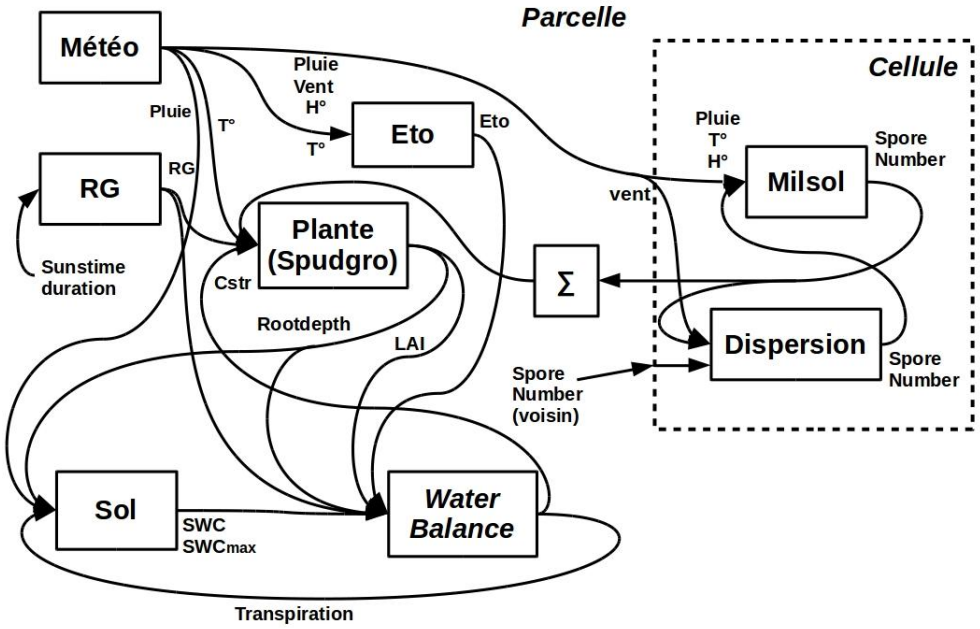


FIGURE 5.16 – Graphe de connexions des modèles atomiques au niveau des cellules et de la parcelle

A partir de l'indice foliaire, de l'évapotranspiration potentielle, du rayonnement et du niveau de la réserve utile, le modèle "Water balance" va estimer l'évapotranspiration réelle. Cette valeur a un rôle fondamental. En effet, elle est l'une des deux grandeurs qui permettent d'estimer la réserve utile du sol et par conséquent de calculer le stress hydrique. Une part importante de nos efforts de modélisation et surtout de paramétrisation, s'est donc portée sur cette grandeur. Par exemple, aux dires d'experts, nous avons réduit la transpiration de la plante lorsque les précipitations sur une journée dépassaient un seuil. Les équations que l'on utilise ne tiennent pas suffisamment compte de l'impact des précipitations. Peut être que l'ajout d'un capteur d'humidité au niveau des feuilles permettrait d'affiner le déclenchement de cette réduction.

Le modèle sol possède une deuxième entrée : la profondeur racinaire. Cette entrée possède un double impact :

— la localisation, dans les couches de sol, du prélèvement d'eau dans la réserve utile. Nous avons adopté comme hypothèse que l'eau était prélevée de manière homogène sur toute la profondeur racinaire.

— le calcul de la réserve utile. On réalise une intégration sur la profondeur racinaire. La même méthode est utilisée pour confronter nos résultats à une mesure plus directe à l'aide de tensiomètres.

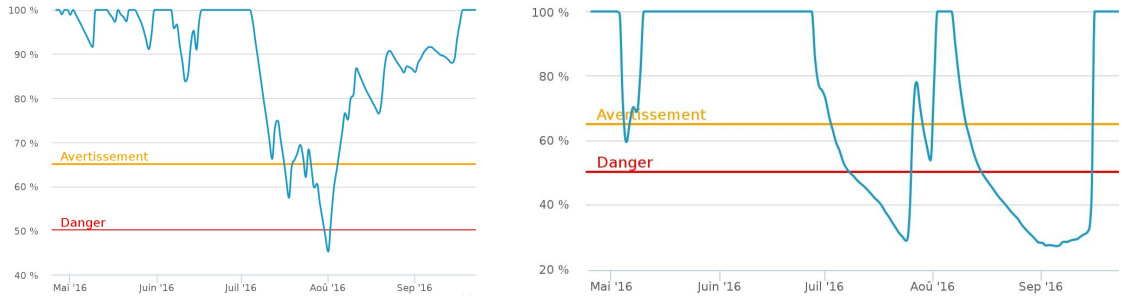


FIGURE 5.17 – Exemple de comparaison de l'estimation de la réserve utile par le modèle à gauche et par les capteurs à droite

La figure 5.17 compare l'estimation de la réserve utile obtenue par le modèle à celle obtenue à l'aide des capteurs. On constate aisément que l'estimation via les modèles sont plus "réactifs" que les capteurs. De nombreuses variations sont "gommées" par les capteurs. C'est totalement logique : le temps de réponse des capteurs est beaucoup plus long. L'atténuation de ce phénomène passe par une recalibration des modèles à l'aide des mesures. Il va falloir probablement lisser la réponse du modèle par une sorte de moyenne mobile. De plus, sur la période mi-août - mi-septembre, les réponses diffèrent. Cela s'explique par la mauvaise prise en compte du modèle de certaines irrigations. C'est la conséquence de la mesure de pluviométrie. Dans ces premières simulations, nous avons utilisé les données brutes des pluviomètres qui sont, en général, installés en bordure de parcelle. Il est donc impératif d'intégrer des informations complémentaires, liées à l'irrigation via des déclarations faites par les agriculteurs. Il est important d'avoir une information de qualité car une fois de plus, l'irrigation est un facteur aggravant pour le mildiou (effet de diffusion par les gouttes d'eau, effet *splashing*).

L'interaction Sol - Transpiration conduit au calcul du stress hydrique ($Cstr$) qui est l'une des nombreuses entrées du modèle Plante. Ce modèle plante permet principalement, dans ce contexte d'utilisation, de calculer la profondeur racinaire et l'indice foliaire (LAI). Néanmoins, toutes les autres variables représentant les différents compartiments (feuille, tige, racine et tubercule) sont calculées. En particulier, il est possible d'estimer le rendement de la récolte. À l'heure actuelle, nous n'avons pas fait de comparaison avec les valeurs réelles car la récolte 2016 n'est pas encore terminée. Ce rendement est impacté par l'effet de la maladie. Pour cela, on impacte l'indice foliaire de la surface de feuille nécrosée par le mildiou. La surface nécrosée est une moyenne sur l'ensemble des cellules formant la parcelle.

$$\begin{aligned} SURFMIL &= SSA * SPORUL \\ PTRi &= \max(0, 1 - e^{-Kdf*(LAI-SURFMIL)}) \end{aligned}$$

Ce retour sur la surface de feuilles nécrosées constitue la rétroaction de la maladie sur la culture. Dans ce modèle, on ne prend pas en compte de potentielles attaques du mildiou par le sol.

5.3.2 Simulation de la propagation du mildiou : exemple de la commune de Oye-Plage

À partir des graphes présentés par les figures 5.15 et 5.16, et du noyau Paradevs, les outils de partitionnement et de création de graphes sont mis en œuvre. Le modèle "Mildiou" nous a permis d'utiliser, à la fois, les aspects de pondération des nœuds et le parallélisme du noyau. Concernant la pondération, nous avons adopté une règle relativement simple puisque le temps de calcul est principalement dû à la phase de dispersion. En effet, ce temps est proportionnel au nombre de voisins et à la superficie des parcelles. Contrairement à nos précédentes simulations, le temps de calcul de chaque modèle n'est pas identique. Il est donc nécessaire de mettre en place un système de pondération des sommets dépendant de la superficie de la parcelle et du nombre de ses voisins. La pondération des arcs est cependant identique dans ce cas. En effet, chaque modèle effectue le même calcul dans le but de déterminer la quantité de spores émises par la fonction de dispersion. Ces modèles sont parfaitement synchrones et ne renvoient que des valeurs numériques, par conséquent il n'est pas nécessaire de mettre en place une pondération. Afin de simplifier les graphes de modèles, la représentation des cellules est simplifiée. Le calcul de la dynamique du mildiou et de sa dispersion est confié à un unique modèle atomique par parcelle. Ce qui conduit à un graphe composé de nœuds fortement liés par paire (interaction plante / mildiou) et où ces paires sont fortement connectées aux paires voisines, en fonction du rayon de propagation. Naturellement, le nombre de voisins est fonction de la distance maximale de dispersion que l'on a défini à partir du modèle météo (direction et puissance du vent) et de la localisation des parcelles. La figure 5.18 illustre un exemple de graphe de modèles issu d'une simulation de propagation de mildiou sur un parcellaire de taille 10. Dans cet exemple, le rayon de propagation du mildiou est relativement faible, ce qui explique le peu de connexions entre des modèles plus éloignés. Cependant, en fonction des paramètres météorologiques il est tout à fait possible d'avoir des connexions entre des sommets spatialement éloignés. Nous avons construit des exemples qui produisent des graphes connexes.

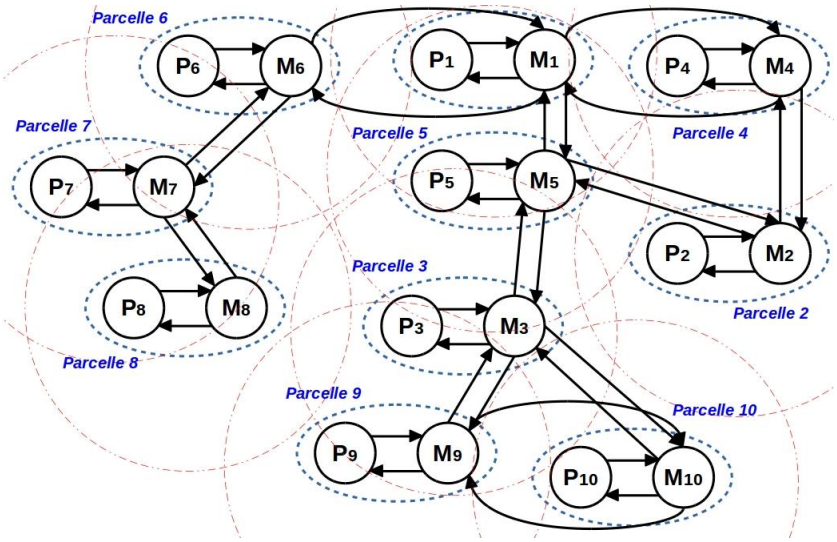


FIGURE 5.18 – Exemple de graphe issu d'une simulation de propagation de mildiou sur un parcellaire de taille 10. Dans cet exemple, la puissance du vent est faible, ce qui explique le faible rayon de contamination.

Dans le graphe de la figure 5.18, chaque modèle de plante P_i est couplé à un modèle de mildiou M_i. Ce modèle couplé représente la i-ème parcelle du parcellaire et est représenté par une ellipse en pointillés de couleur bleue. Les parcelles sont connectées les unes aux autres par le biais des modèles M_i suivant un rayon de propagation de la maladie (représenté par des cercles rouges en pointillé très fins). Tout modèle M_j appartenant à un cercle de contamination de la parcelle i se voit connecté aux modèles M_i. Le rayon du cercle est dépendant de la météo, principalement de la puissance du vent fournie par le modèle de plante P_i (contenue dans le sous-modèle météo). Nous parlions tout à l'heure d'une discrétisation des parcelles en cellules de taille 20m $\times$ 20m, cette dernière est contenue par les états des modèles M_i. En effet, chaque M_i contient autant d'états que la parcelle contient de cellules.

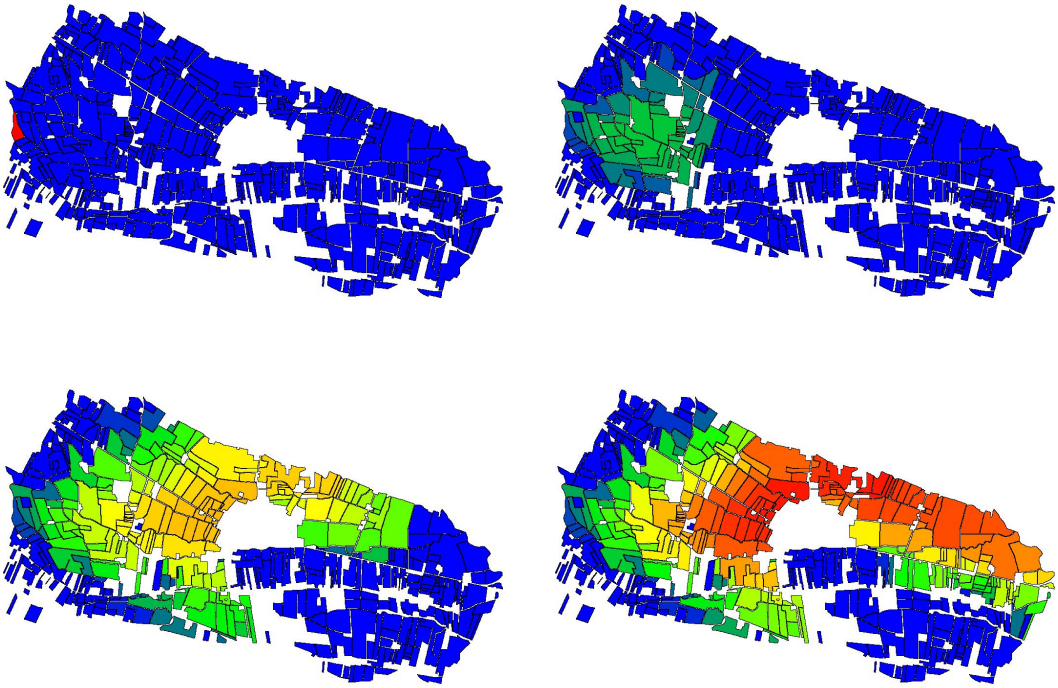


FIGURE 5.19 – Propagation du mildiou : de la situation initiale à t = 7 jours

La figure 5.19 montre la propagation du mildiou sur un exemple où l'ensemble des parcelles accueillent des pommes de terre et où le vent dominant est orienté ouest. On peut clairement observer la propagation à l'aide d'un arc-en-ciel de couleurs, allant du bleu (parcelle seine) au rouge (parcelle très contaminée). En effet à $t = 0$, le niveau de concentration de spores infectieuses est très élevé sur une unique parcelle située plein ouest, le reste du parcellaire est sain du fait que la dispersion n'est pas encore eu lieu. Cette couleur implique une très forte contamination de la parcelle avec un risque de perte de rendement quasi-totale si l'infection a lieu au début du cycle de croissance de la plante. À $t = 3$ jours, nous pouvons constater une propagation des spores, avec un niveau de contamination plus faible. En effet, le nombre de spores émises étant supposé homogène, il est normal qu'au vu du nombre de parcelles contaminées, la quantité de spores par parcelle soit moins importante que sur la parcelle émettrice. De plus, nous pouvons

observer que la parcelle émettrice à $t = 0$ est redevenue bleue, ce qui implique un nombre de spores infectieuses faibles. En effet, comme le montre le schéma de la figure 5.3, les spores ne sont pas infectieuses durant toute la durée du cycle. Il est donc normal que la couleur des parcelles tendent vers le bleu après contamination. En observant les deux autres images, nous pouvons constater une amplification de la contamination du parcellaire. En effet à $t = 7$ jours, une majeure partie du territoire se voit contaminée, à des taux relativement importants. Cet exemple illustre parfaitement le caractère explosif de la propagation du mildiou dans des conditions favorables. En effet, moins d'une semaine suffit à contaminer près de 2000 hectares de cultures. La perte de rendement restera cependant liée à la période du cycle de croissance de la plante durant laquelle la contamination a lieu.

Naturellement, ces cartes sont loin de refléter la réalité, du fait qu'il manque plusieurs aspects importants. Premièrement, on considère qu'aucun traitement fongicide n'est appliqué, ce qui dans la réalité est totalement faux. Le coût d'une perte de rendement étant tellement plus préjudiciable que le coût de traitement, les agriculteurs ne prennent pas le moindre risque. Deuxièmement, nous avons déclaré un seul foyer initial de contamination. Or en réalité, les sources sont multiples : tas de déchets non bâchés (non obligatoire en France), repousses dans les parcelles, cultures précoces de pommes de terre dans les potagers des particuliers, manque de traitement dans ces derniers, etc ... De plus, la présence de barrières naturelles, telles que les forêts ou les villes, sont autant de facteurs limitants à la prolifération du mildiou qui ne sont actuellement pas pris en compte dans notre modèle. Le couplage de ces informations de terrain, aux modèles précédemment présentés aurait pour effet de fournir une simulation plus proche de la réalité. Cependant, il n'existe pas de moyen fiable pour garantir l'exactitude de ces informations. Elle repose sur la collaboration entre les agriculteurs et les particuliers producteurs de pommes de terre.

Conclusion

La modélisation de la propagation du mildiou n'est pas chose facile. En effet, même si les modèles de plante et de contamination existent depuis plusieurs dizaines d'années, il est difficile de les paramétrer efficacement. L'une des grandes nouveautés de cette thèse, est de proposer une météo géolocalisée basée sur l'utilisation de capteurs à même le champ. En effet, l'utilisation de données locales, contrairement aux données globales utilisées dans les autres modèles, permet un paramétrage précis de nos modèles. La plante, ainsi que le cycle de génération de spores sont tous deux influencés par les caractères climatiques (température, pluviométrie, hydrométrie et vent). L'utilisation d'objets connectés ajoute un aspect temps réel à l'obtention des données, ce qui peut permettre d'effectuer un recalibrage des modèles dans le but de garantir leur convergence.

Notre modélisation de la dynamique du mildiou, via la contamination aérienne, repose sur trois modèles principaux : le modèle de plante (Spudgro), le modèle de développement des spores (Milsol) et un modèle de dispersion. Ces trois modèles nécessitent un certain nombre de paramètres indispensables à leur fonctionnement. Contrairement à certaines modélisations qui utilisent une paramétrisation globale, nous avons fait le choix de coupler ces modèles à d'autres modèles complémentaires telles que : la météo géolocalisée, la gestion hydrique du sol ou encore la transpiration de la plante. En effet, les données fournies par ces modèles sont en interaction directe

avec la dynamique du mildiou *P. infestant*. Une bonne connaissance de l'état du sol et des besoins en eau de la plante sont des informations primordiales pour minimiser les risques d'épidémie.

Comme nous l'avons vu dans la section 5.3.1, le nombre de modèles ainsi que leurs dépendances sont importants. Cependant, il est possible de complexifier davantage cette modélisation. En effet, dans notre cas, nous ne gérons que la partie aérienne de la contamination. Or, ce n'est pas le seul facteur de contamination, le sol en fait également partie. Cependant, la contamination par les airs étant le principal facteur de prolifération, nous avons fait le choix de ne nous intéresser qu'à ce dernier. Les résultats présentés par notre première version du modèle de prolifération du mildiou sont tout à fait encourageants. Cependant, le modèle est encore trop "pauvre" pour garantir une fiabilité totale. En effet, bon nombre d'informations qualitatives ne sont pas prises en compte au cours de la simulation. C'est notamment le cas des traitements fongicides réalisés, de la localisation des tas de pomme de terre (non bâchés), ou encore la présence de cultures de pommes de terre chez les particuliers. Il est important de souligner que ces cultures, même à très petites échelles, sont les premiers foyers d'infection, généralement par manque de traitement. Cependant, toutes ces informations sont difficiles à obtenir et ne sont pas toujours d'une fiabilité totale. Actuellement, l'outil de simulation tel qu'il est conçu, reste uniquement un outil d'aide à la décision globale, à condition qu'un maximum d'informations soit mis à disposition du modélisateur pour le paramétrage des modèles (localisation des tas de pommes de terre, par exemple).

La précision fournie par les modèles Spudgro (plante) et Milsol (mildiou) dépend fortement du niveau de rigueur apporté aux sous-modèles en amont. Il existe une quantité quasi-infinie de modélisations différentes d'un même problème, seul le niveau de précision varie. Dans notre cas, nous avons fait le choix d'ajouter un modèle de sol, sans pour autant tenir compte des éventuels croisements génétiques pouvant avoir lieu au cours des dispersions. Cette remarque met en avant toutes les perspectives qui s'offrent à nous concernant les améliorations futures du modèle mildiou. Le travail conjoint des agronomes, des agriculteurs et des modélisateurs semble être une condition *sinequanone* au développement d'une agriculture éco-responsable, garantissant une gestion durable du mildiou de la pomme de terre à grande échelle.

Conclusion générale et perspectives

L'augmentation incessante de la taille des simulations en nombre de modèles et en besoin mémoire, limite l'utilisation de la simulation parallèle. En effet, face à des calculs trop gourmands en mémoire, un processeur seul ne suffit pas toujours. Cette problématique pousse la communauté DEVS à s'orienter vers la simulation distribuée. Même si cette dernière offre une perspective presque illimitée en terme de mémoire et de nombre de modèles, elle peut être cependant freinée par des temps de simulation trop grands. En effet, comme le disent les adages *"le temps c'est de l'argent"* et "*l'argent est le nerf de la guerre"*, il est important d'optimiser la distribution des modèles de façon à diminuer, le plus possible, le temps d'une simulation. Dans cette thèse, nous proposons une optimisation des temps de simulation, par une restructuration de la hiérarchie de modèles, de façon à la rendre optimale pour la distribution. Cette optimalité repose sur un équilibre de charge entre les différents nœuds de calcul et sur une minimisation du nombre de transferts d'événements entre eux. Cette problématique est facilement assimilable à un problème de partitionnement de graphes, où chaque sommet correspond à un modèle atomique et chaque arc symbolise les échanges d'événements entre les modèles. En effet, la mise à plat de la hiérarchie permet d'obtenir une vue globale de la simulation, sous la forme d'un graphe de modèles. Le partitionnement de ce dernier, de façon à respecter la problématique précédemment citée, permet de garantir une optimalité de la distribution.

Cependant, ce partitionnement ayant lieu avant le lancement de la simulation, il est difficile de prévoir le comportement des modèles. Comme nous le savons, un modèle atomique DEVS peut être soumis à des entrées. Celles-ci peuvent modifier la dynamique du modèle et donc faire considérablement varier la fréquence d'événements en transit sur le réseau. Or, l'utilisation du réseau engendre un surcoût de communication. Il est donc très important, contrairement à l'approche parallèle, de minimiser le plus possible l'utilisation de ce dernier. Pour que cette information soit prise en compte, il est nécessaire de pondérer le graphe de modèles de façon à guider le partitionnement. Dans cette thèse, nous proposons une démarche d'apprentissage des dynamiques à partir des chaînes de Markov cachées. Cependant, l'utilisation de ces dernières nécessite quelques modifications liées au formalisme DEVS. Cette démarche n'a d'intérêt que si la simulation se compose d'un graphe de modèles où les dynamiques sont fortement hétérogènes, c'est à dire que la fréquence d'émission d'événements est très variable en fonction de la localisation des modèles dans le graphe. L'utilisation de nos exemples (dynamiques artificielles pour le benchmark) n'a malheureusement pas permis de mettre en avant toute l'importance de cette démarche dans le cadre d'une distribution. Cependant, en effectuant une simple comparaison des résultats de partitionnement,

nous avons pu constater que la pondération joue un rôle important sur la qualité de cette dernière. L'analyse des paramètres de la simulation semble être une piste intéressante pour déterminer l'intérêt d'une éventuelle pondération. En effet, au vu du prix de l'apprentissage, il est nécessaire de garantir l'impact que peut avoir la pondération du graphe de modèles sur la qualité de la distribution. Une étude approfondie des t_a de chaque modèle, ainsi que de la structure du graphe pourrait être une piste envisageable, à l'image d'une classification de simulations en fonction de leurs paramètres. La mise en place d'un processus d'apprentissage hybride, liant simulation et chaînes de Markov cachées, pourrait être une perspective intéressante dans le but d'améliorer un peu plus la qualité de la pondération. Cette vision de l'apprentissage aurait pour intérêt majeur de garantir la fiabilité des entrées des modèles appris.

Malheureusement, l'optimalité de la distribution n'est pas exclusivement dépendante de la qualité de la partition. En effet, le noyau de simulation, tel qu'il est conçu actuellement, n'offre pas d'algorithme tel que *null-message* ou *Time wrap*. On parle alors de simulation distribuée pessimiste. Dans ce cas, lorsque les modèles sont parfaitement synchrones, le gain obtenu tend vers le gain théorique maximum. Cependant, dans la réalité du monde physique, les modèles ont plutôt tendance à avoir des comportements asynchrones. D'un point de vue DEVS, la synchronisation des modèles repose sur des t_a équivalents et de préférence entiers. Cependant, il est rare que chaque modèle atomique se compose d'un même t_a, et qui plus est entier. Dans ce cas, les résultats de simulation distribuée sont de moins bonne qualité, sans pour autant être totalement catastrophiques. En effet, le peu de gain réalisé par la distribution des modèles est conservé par une minimisation des coûts de transfert. La gestion des simulations aux modèles asynchrones, ou partiellement asynchrones, nous pousse à envisager d'autres stratégies d'implémentation du noyau de simulation. En effet, la contrainte de non utilisation d'algorithmes en approche optimiste est un frein trop important dans ce cas. L'implémentation d'un noyau de simulation optimiste semble être une solution viable pour améliorer nos performances. En effet, dans ce cas, la simulation n'est pas contrainte à respecter l'ordre total d'exécution des modèles. Tout modèle se situant dans une petite fenêtre temporelle (lookhead) est autorisé à effectuer son exécution, au risque de devoir effectuer un retour en arrière (rollback) si ce dernier nécessite une entrée n'ayant pas encore eu lieu lors de son exécution. Cette stratégie permet d'élargir la taille des ensembles IMM (exécution imminente) et permet, par conséquent, une exécution simultanée d'un nombre plus important de modèles. Cette stratégie d'implémentation est payante à condition que la fenêtre temporelle soit bien définie, c'est à dire que le nombre de rollbacks ne soit pas trop important.

Cette stratégie ne constitue pas la seule perspective d'améliorations de notre démarche. En effet, il est important de rappeler qu'un modèle DEVS peut avoir beaucoup de comportements différents. Lorsque les modèles qui composent une simulation sont passifs, c'est à dire qu'ils possèdent un t_a infini et qu'ils restent en attente tant qu'aucun événement extérieur ne vient le perturber, la simulation est parfaitement asynchrone. Dans ce cas, le passage à une simulation optimiste n'aurait pas le moindre effet. En effet, un modèle effectue son exécution à condition que chacun de ses parents ait effectué la sienne. Il y a donc un lien de dépendance très important entre les modèles. Pour cette catégorie de simulations, il serait intéressant d'envisager une autre approche de partitionnement, reposant exclusivement sur le lien de dépendance des modèles. La

mise en couches de ces derniers en fonction de leur "date" activation permettrait de les distribuer plus efficacement dans ce cas. Cette perspective a l'avantage d'équilibrer au mieux la charge de calcul sur un cas complexe, mais ne permet pas de minimiser le coût de transfert. La prise en compte de cette seconde contrainte reste une piste à exploiter.

L'application d'un modèle de propagation du mildiou sur notre noyau de simulation optimisé pour la distribution (Paradevs) permet de traiter des échelles spatiales beaucoup plus importantes. En effet, les modèles mildiou étaient initialement appliqués à l'échelle de la parcelle, alors qu'actuellement, il est possible d'en simuler la prolifération à l'échelle du territoire, voire même de la région. Le tout étant de disposer des informations nécessaires à son exécution (comme la météo par exemple). En effet, outre l'augmentation de l'échelle spatiale de traitement, cette thèse a permis d'apporter une autre vision de la modélisation du mildiou. L'apport d'une technologie connectée, à base de capteurs géolocalisés, pour récolter différentes informations météorologiques et quantitatives a permis de mettre en place un modèle météo offrant des données plus fiables et plus fréquentes (une toutes les quinze minutes). De plus, le couplage des modèles de plante et de mildiou (génération + dispersion) à des modèles de gestion du sol et de la transpiration permettent d'obtenir des résultats beaucoup plus fiables. Cette remarque ouvre une perspective majeure concernant le développement d'un modèle mildiou complet. En effet, actuellement le modèle n'est conçu que pour traiter l'aspect aérien de la contamination. Or, la contamination peut également avoir lieu par le sol, lorsque les spores retombent à la surface, à l'aide des eaux de pluies. De plus, la dynamique qui anime le mildiou est en réalité bien plus complexe que ce qu'on peut imaginer. Elle ne dépend pas que de paramètres physiques mais aussi de données qualitatives à l'image des traitements fongicides, de la présence de tas de pomme de terre, etc ... La création d'un modèle complet repose donc sur une collaboration étroite des agronomes, des agriculteurs et des modélisateurs, pour répondre aux besoins de l'agriculture de demain : l'agriculture 2.0.

Bibliographie

[APHF+05] J. L. Andrade-Piedra, R. J. Hijmans, G. A. Forbes, W. E. Fry, and R. J. Nelson. Simulation of potato late blight in the andes. i : Modification and parameterization of the lateblight model. *Phytopathology*, 95(10) :1191–1199, 2005. 172

[Bar98] F. J. Barros. Abstract simulators for the dsde formalism. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 407–412. IEEE, 1998. 1

[BC94] P. Baldi and Y. Chauvin. Smooth on-line learning algorithms for hidden markov models. *Neural Computation*, 6(2) :307–318, 1994. 92

[BCG+13] J.-E. Bergez, P. Chabrier, C. Gary, M.H. Jeuffroy, D. Makowski, G. Quesnel, E. Ramat, H. Raynal, N. Rousse, D. Wallach, P. Debaeke, P. Durand, M. Duru, J. Dury, P. Faverdin, C. Gascuel-Odoux, and F. Garcia. An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems. *Environmental Modelling & Software*, 39 :39 – 49", 2013. 161

[Bee52] A. Beer. Bestimmung der absorption des rothen lichts in farbigen flüssigkeiten [determination of the absorption of red light in colored liquids]. *Annalen der Physik*, 162(5) :78–88, 1852. 171

[BF96] Y. Bengio and P. Frasconi. Input-output hmms for sequence processing. *IEEE Transactions on Neural Networks*, 7(5) :1231–1249, Sep 1996. 88

[Bic07] C-E. Bichot. A new method, the fusion fission, for the relaxed k-way graph partitioning problem, and comparisons with some multilevel algorithms. *Journal of Mathematical Modeling and Algorithms (JMMA)*, pages 319–344, 2007. 32

[Bil98] J. A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *University of California Berkeley*, 1998. 92

[BJ93] T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In *Parallel Processing for Scientific Computing*, pages 445–452, 1993. 43

[Bry77] R. E. Bryant. Simulation of packet communication architecture computer systems. 1977. 19

[BS94] S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Practice and Experience*, 6(2) :101–117, 1994. 43

[BS10] C.E. Bichot and P. Siarry. *Partitionnement de graphe, Optimisation et applications*. Hermes Science Publications, 2010. 48

[BS13] C. E. Bichot and P. Siarry. *A Partitioning Requiring Rapidity and Quality : The Multilevel Method and Partitions Refinement Algorithms*, pages 27–63. John Wiley & Sons, Inc., 2013. 34

[BSVdB97] T. Brouard, M. Slimane, G. Venturini, and J.P. Asselin de Beauville. Apprentissage du nombre d'états d'une chaîne de markov cachée pour la reconnaissance d'images. *SEIZIÈME COLLOQUE GRETSI*, pages 845–848, 1997. 99

[CHC12] X. Cui, J. Huang, and J. T. Chien. Multi-view and multi-objective semi-supervised learning for hmm-based automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(7) :1923–1935, Sept 2012. 88

[Cho96] A. C.-H. Chow. Parallel devs : A parallel, hierarchical, modular modeling formalism and its distributed simulator. *Trans. Soc. Comput. Simul. Int.*, 13(2) :55–67, December 1996. 12

[CM79] K. M. Chandy and J. Misra. Distributed simulation : A case study in design and verification of distributed programs. *IEEE Transactions on software engineering*, (5) :440–452, 1979. 19

[CM81] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4) :198–206, 1981. 19

[CM10] A. Cornuéjols and L. Miclet. *Apprentissage artificiel : Concepts et algorithmes*. Eyrolles, 2010. 87

[CZ94] A. C. H. Chow and B. P. Zeigler. Parallel devs : A parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th Conference on Winter Simulation*, WSC '94, pages 716–722, San Diego, CA, USA, 1994. Society for Computer Simulation International. 1, 12

[DCG99] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2) :137 – 172, 1999. 34

[DGK04] I. S. Dhillon, Y. Guan, and B. Kullis. A unified view of kernel k-means, spectral clustering and graph partitioning. *Rapport technique TR-04-25, University of Texas at Austin*, 2004. 32, 37

[DGK07] I. S. Dhillon, Y. Guan, and B. Kullis. Weighted graph cuts without eigenvectors : A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. 32

[DH72] W.E. Donath and A.J. Hoffman. Algorithms for partitioning graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15(3) :938–944, 1972. 36

[DPST03] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. Métaheuristiques pour l'optimisation difficile. *Eyrolles*, 2003. 34

[ESYD03] A. Elgammal, V. Shet, Y. Yacoob, and L. S. Davis. Learning dynamics for exemplar-based gesture recognition. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–571–I–578 vol.1, June 2003. 88

[FH04] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, pages 167–181, 2004. 30

[Fie75] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4) :619–633, 1975. 37

[FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181, June 1982. 48

[Fuj87] R. M. Fujimoto. Performance measurements of distributed simulation strategies. Technical report, DTIC Document, 1987. 18

[Fuj00] R. M. Fujimoto. *Parallel and distributed simulation systems*, volume 300. John Wiley and Sons, 2000. 18

[GF05] Yuli Gao and Jianping Fan. Semantic image classification with hierarchical feature subset selection. In *Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval*, MIR '05, pages 135–142, New York, NY, USA, 2005. ACM. 33

[GL96] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996. 37

[GW05] E. Glinsky and G. Wainer. Devstone : a benchmarking technique for studying performance of devs. modeling and simulation environments. *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings*, pages 265–272, 2005. 108

[HAJ90] Xuedong D Huang, Yasuo Ariki, and Mervyn A Jack. *Hidden Markov models for speech recognition*, volume 2004. Edinburgh university press Edinburgh, 1990. 91

[HK92] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9) :1074–1085, Sep 1992. 37

[HL95] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.*, 16(2) :452–469, March 1995. 37

[Hol81] I. Holyer. The np-completness of some edge-partition problems. *SIAM Journal of Computing*, pages 713–717, 1981. 26

[Hor77] T. W. Horst. A surface depletion model for deposition from a gaussian plume. *Atmospheric Environment (1967)*, 11(1) :41 – 46, 1977. 166

[HQR15a] C. Herbez, G. Quesnel, and E. Ramat. Building partitioning graphs in parallel-devs context for parallel simulations. In *Proceedings of the 2015 Spring Simulation Conference*, 2015. 30

[HQR15b] C. Herbez, G. Quesnel, and E. Ramat. Optimization of parallel-devs simulations with partitioning techniques. In *Proceedings of the 2015 Simultech Conference*, 2015. 81

[HQR15c] C. Herbez, G. Quesnel, and E. Ramat. Optimizing distributed devs simulations with partitioning and hidden markov model learning methods. In *The 2015 european simulation and modelling conference, Modelling and Simulation '2015*, pages 133 – 140, 2015. 82

[HQR16] C. Herbez, G. Quesnel, and E. Ramat. Optimisation des simulations devs distribuées par apprentissage indépendant de la dynamique des modèles. In *JDF 2016 - LES JOURNÉES DEVS FRANCOPHONES - THÉORIE ET APPLICATIONS*, 2016. 82

[Jef85] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3) :404–425, 1985. 20

[JJT86] KB Johnson, SB Johnson, and PS Teng. Development of a simple potato growth model for use in crop-pest management. *Agricultural systems*, 19(3) :189–209, 1986. 168

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598) :671–680, 1983. 34

[KH95] P. L. Kooman and A. J. Haverkort. *Modelling development and growth of the potato crop influenced by temperature and daylength : LINTUL-POTATO*, pages 41 – 59. Springer Netherlands, 1995. 168

[KK95] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM. 47

[KK98a] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1) :359–392, December 1998. 41, 42, 47

[KK98b] G. Karypis and V. Kumar. Metis : A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota*, 1998. 47

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2) :291–307, 1970. 29, 47

[KLG+02] P. Knight, A. Corder R. Liedel, J. Giddens, R. Drake, C. Jenkins, and P. Agarwal. Evaluation of run time infrastructure (rti) implementations. In *Huntsville Simulation Conference*, 2002. 18

[KMN+02] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm : analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 881 – 892, 2002. 33

[Lad88] L. Ladeveze. Présentation du modèle milsol. *Rapport interne*, 1988. 172

[Lam60] J. Lambert. Photometria, sive de mensura et gradibus luminis, colorum et umbrae (augsberg : Eberhard klett) [photometry, or, on the measure and gradations of light, colors, and shade]. page 391, 1760. 171

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978. 19

[Pat] Fao penman-monteith equation. http://www.fao.org/docrep/x0490e/x0490e06.htm#chapter. 180

[PF98] P. Perona and W. Freeman. A factorization approach to grouping. *In Proceedings of the European Conference on Computer Vision*, 1406, 1998. 30

[PSL90] A. Pothen, H. D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3) :430–452, 1990. 37

[QDR09] G. Quesnel, R. Duboz, and É. Ramat. The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17 :641–653, April 2009. 124

[Rab63] M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3) :230 – 245, 1963. 86

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected application in speech recognition. *Proceeding of the IEEE*, 77(2) :257–286, 1989. 88

[Rad] Meteorological data. http://www.fao.org/docrep/x0490e/x0490e07.htm#radiation. 180

[Rak12] T. F. Rakotonindraina. Analyse et modélisation des effets des pratiques culturales sur les épidémies de mildiou de la pomme de terre. adaptation du modèle sippom au pathosystème. *Rapport de thèse*, 2012. vii, 161, 166, 168, 170, 177

[RWF93] R. Raposo, D. S. Wilks, and W. E. Fry. Evaluation of potato late blight forecasts modified to include weather forecasts : a simulation analysis. *Phytopathology*, 83(1) :103–108, 1993. 172

[Sam85] B. Samadi. Distributed simulation, algorithms and performance analysis (load balancing, distributed processing). 1985. 20

[Sch96] H. Scherm. On the velocity of epidemic waves in model plant disease epidemics. *Ecological Modelling*, 87(1) :217 – 222, 1996. 166

[SD89] P. Siarry and G. Dreyfus. La méthode du recuit simulé : théorie et applications. *ESPCI-IDSET*, 1989. 34

[SJ00] J. Shi and J.Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 888–905, 2000. 29

[SSVP02] L. Serradura, M. Slimane, N. Vincent, and C. Proust. Classification semi-automatique de documents web à l'aide des chaînes de markov cachées. *actes de INFORSID*, pages 215–228, 2002. 88

[ST97] H. D. Simon and SH. Teng. How good is recursive bisection ? *SIAM Journal on Scientific Computing*, pages 1436 – 1445, 1997. 28

[TKP+02] D. J. Timlin, S. H. Kim, Ya. Pachepsky, V. R. Reddy, C. Fraisse, A. Alva, and J. T. Baker. 2dspud, a two-dimensional model of potato growth and development. In *ASA-CSSA-SSSA Proceedings*, number 336. ASA-CSSA-SSSA Proceedings. Poster, 2002. 168

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2) :260–269, April 1967. 91

[WC89] Y.-C. Wei and C.-K. Cheng. Towards efficient hierarchical designs by ratio cut partitioning. In *Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on*, pages 298–301, Nov 1989. 29, 30

[WG01] G. Wainer and N. Giambiasi. Application of the cell-devs paradigm for cell spaces modelling and simulation. *Simulation*, 76(1) :22–39, 2001. 1

[Wil12] A. Williams. *C++ concurrency in action : Practical Multithreading*. Manning Publications, 2012. 126

[YOI92] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 379–385, Jun 1992. 88

[ZCLS98] B. P. Zeigler, H. Cho, J. Lee, and H. Sarjoughian. The devs/hla distributed simulation environment and its support for predictive filtering. *DARPA Contract N6133997K-0007 : ECE Dept., UA, Tucson, AZ*, 1998. 18

[Zei73] B.P. Zeigler. A conceptual basis for modeling and simulation. *SiMULETTER*, 5(1) :23 – 32, 1973. 6

[Zei76] B. P. Zeigler. *Theory of Modeling and Simulation*. Wiley Interscience, 1976. 1, 8

[Zei84] B. P. Zeigler. *Theory of Modeling and Simulation*. Krieger Publishing Company, 2nd edition, 1984. 11

[ZKP00] B. P. Zeigler, D. Kim, and H. Praehofer. *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2nd edition, 2000. 14, 16

[ZL98] B. P. Zeigler and J. S. Lee. Theory of quantized systems : formal basis for devs/hla distributed simulation environment. In *Aerospace/Defense Sensing and Controls*, pages 49–58. International Society for Optics and Photonics, 1998. 18

Parallélisation massive de dynamiques spatiales : contribution à la gestion durable du mildiou de la pomme de terre

La simulation à événements discrets, dans le contexte du formalisme DEVS, est en plein essor depuis quelques années. Face à une demande grandissante en terme de taille de modèles et par conséquent en temps de calcul, il est indispensable de construire des outils tel qu'ils garantissent une optimalité ou au mieux une excellente réponse en terme de temps de simulation. Certes, des outils de parallélisation et de distribution tel que PDEVS existent, mais la répartition des modèles au sein des nœuds de calculs reste entièrement à la charge du modélisateur.

L'objectif de cette thèse est de proposer une démarche d'optimisation des temps de simulation parallèle et distribuée, en restructurant la hiérarchie de modèles. La nouvelle hiérarchie ainsi créée doit garantir une exécution simultanée d'un maximum de modèles atomiques, tout en minimisant le nombre d'échanges entre modèles n'appartenant pas au même noeud de calculs (i.e. au même sous-modèle). En effet, l'optimisation des temps de simulation passe par une exécution simultanée d'un maximum de modèles atomiques, mais dans un contexte distribué, il est important de minimiser le transfert d'événements via le réseau pour éviter les surcoûts liés à son utilisation. Il existe différentes façons de structurer un modèle DEVS : certains utilisent une structure hiérarchique à plusieurs niveaux et d'autres optent pour une structure dite "à plat". Notre approche s'appuie sur cette dernière. En effet, il est possible d'obtenir un unique graphe de modèles, correspondant au réseau de connexions qui lient l'ensemble des modèles atomiques. A partir de ce graphe, la création d'une hiérarchie de modèles optimisée pour la simulation distribuée repose sur le partitionnement de ce graphe de modèles. En effet, la théorie des graphes offre un certain nombre d'outils permettant de partitionner un graphe de façon à satisfaire certaines contraintes. Dans notre cas, la partition de modèles obtenue doit être équilibrée en charge de calcul et doit minimiser le transfert de messages entre les sous-modèles.

L'objectif de cette thèse est de présenter la démarche d'optimisation, ainsi que les outils de partitionnement et d'apprentissage utilisés pour y parvenir. En effet, le graphe de modèles fournit par la structure à plat ne contient pas toutes les informations nécessaires au partitionnement. C'est pourquoi, il est nécessaire de mettre en place une pondération de celui qui reflète au mieux la dynamique individuelle des modèles qui le compose. Cette pondération est obtenue par apprentissage, à l'aide de chaînes de Markov cachées (HMM). L'utilisation de l'apprentissage dans un contexte DEVS a nécessité quelques modifications pour prendre en compte toutes ces spécificités. Cette thèse présente également toute une phase de validation : à la fois, dans un contexte parallèle dans le but de valider le comportement du noyau de simulation et d'observer les limites liées au comportement des modèles atomiques, et d'autre part, dans un contexte distribué. Pour terminer, cette thèse présente un aspect applicatif lié à la gestion durable du mildiou de la pomme de terre. Le modèle mildiou actuel est conçu pour être utilisé à l'échelle de la parcelle. En collaboration avec des agronomes, nous proposons d'apporter quelques modifications à ce dernier pour étendre son champs d'action et proposer une nouvelle échelle spatiale.

Mots clés : Simulation distribuée, DEVS, Partitionnement, Apprentissage, Modélisation, Mildiou

Massive scale parallelization of spatial dynamics : Input for potato blight sustainable management

Discrete-event simulation, in a context of DEVS formalism, has been experiencing a boom over the recent years. In a situation of increasing demand in terms of model size and consequently in calculation time, it is necessary to build up tools to ensure optimality, or even better, an excellent response to simulation times. Admittedly, there exist parallelization and distribution tools like PDEVS, but the distribution of models within compute nodes is under the modeler's sole responsibility.

The Ph.D. main scope is to propose an optimization approach of parallel and distributed simulation times, restructuring the hierarchy of models. The new founded hierarchy can thus guarantee a simultaneous execution of a maximum quantity of atomic models while minimizing the number of exchanges between models, which are not associated with the same calculation node (i.e. with the same sub-model). Accordingly, optimizing simulation time goes through a simultaneous implementation of a maximum quantity of atomic models, but in a distributed context it is highly important to minimize the adaptation transfer via the network to avoid overcharges related to its use. There exist different ways of structuring a DEVS model : some scientists use a multi-leveled hierarchical structure, and others opt for a "flat" structure. Our objective focuses on the latter. Indeed, it is possible to obtain a single graph of models, corresponding to the connection network linking all the atomic models. From this graph the creation of a model hierarchy optimized by the distributed simulation focuses on the partitioning of this model graph. In such cases, the graph theory reveals a certain number of tools to partition the graph to meet some constraints. In our study, the resulting model partition must not only balance calculation needs but also minimize the message transfer between sub-models.

The Ph.D. main scope is to propose not only an optimization approach but also partitioning and learning tools to achieve full compliance in our processing methods. In such cases, the model graph using the flat structure does not provide us with all the necessary information related to partitioning. That is the reason why it is highly necessary to assign a weighting in the graph that best reflects the individual dynamics of models composing it. This weighting comes from learning, using the Hidden Markov Models (HMM). The use of learning in DEVS context results in some adjustments to consider all the specificities. The thesis also ensures the complete validation phase, either in a parallel context to validate the simulation node behavior and observe the limits of atomic model behavior, or in a distributed context. This dissertation in its final state also includes a practice-oriented approach to sustainably manage potato blight. The current fungus Phytophthora infestans simulation model is conceived for a plot scale. In collaboration with agronomists, we provide a few changes to update the Phytophthora infestans model to extend the scope of action and propose a new scale of values.

Key words : Distributed simulation, DEVS, Partitioning, Learning, Modelisation, Potato blight.