Model-checking pour l'agriculture de précision

Rim Saddem

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Génie informatique, automatique et traitement du signal

École doctorale I2S

Unité de recherche LIRMM, UMR 5506

Model-checking pour l'agriculture de précision

Présentée par Rim SADDEM
Le 20/06/2019

Sous la direction de Didier Crestani,
Olivier Naud et Karen Godary-Dejean

Devant le jury composé de

Roland Lenain ,	DR - IRSTEA Clermont-Ferrand	Examinateur
Frédéric Garcia,	DR - INRA - INRA Toulouse	Rapporteur
Laurent Pietrac,	MC HdR - INSA Lyon	Rapporteur
Didier Crestani,	Professeur - IUT Montpellier-Sète/UM	Directeur de thèse
Olivier Naud,	Ingénieur de Recherche - IRSTEA Montpellier	Encadrant
Karen Godary-Dejean,	MC - Polytech Montpellier/UM	Encadrante

UNIVERSITÉ DE MONTPELLIER

Remerciements

Tout d'abord, je remercie ***Dieu*** *de m'avoir donné la patience nécessaire pour la production de ce travail.*

Ces travaux n'auraient pu arriver à terme sans l'aide de tous ceux qui ont contribué à leur succès.

Je remercie Monsieur ***Roland LENAIN****, Directeur de Recherche à IRSTEA de Clermont-Ferrand, pour avoir accepté de présider le jury de la soutenance.*

J'adresse tous mes remerciements à Monsieur ***Laurent PIETRAC****, maître de conférences HDR à l'INSA de Lyon, ainsi qu'à Monsieur* ***Frédéric GARCIA****, directeur de recherche à l'INRA de Toulouse, de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse.*

Je tiens à exprimer mon respect, ma grande gratitude et mes vifs remerciements à mon cher directeur de thèse Monsieur ***Didier CRESTANI****, professeur à l'IUT de Montpellier-Sète/UM, pour son encadrement tout au long de cette thèse, ses précieux conseils, sa générosité et pour les nombreux encouragements qu'il m'a prodigués. Son soutien moral et son aide précieuse m'ont permis d'effectuer cette thèse dans les meilleures conditions. C'est également à ses côtés que j'ai appris les bases de l'enseignement.*

Je tiens également à exprimer mon respect, ma profonde reconnaissante et mes remerciements à ma chère encadrante Madame ***Karen GODARY-DEJEAN****, maître de conférences à Polytech Montpellier/UM, pour m'avoir apporté sa connaissance et un soutien permanent dans ma thèse. Elle a consacré son précieux temps à me prodiguer de nombreux conseils, à me former et m'accompagner tout au long de ces 4 ans d'expérience professionnelle. C'est un grand plaisir de travailler avec une personne non seulement experte en vérification formelle mais aussi une personne dotée de qualités humaines exceptionnelles.*

Je voudrais remercier du plus profond de mon coeur mon cher encadrant Monsieur ***Olivier NAUD****, Ingénieur de recherche à IRSTEA de Montpellier, pour son génial encadrement, sa grande gentillesse, sa patience, sa disponibilité permanente, ses judicieux conseils et surtout pour son aide précieuse pour ma recherche bibliographique. Il a partagé, avec moi, ses intuitions, ses idées, son savoir et sa connaissance. C'est grâce à lui que j'ai appris la rigueur scientifique et la précision. Je suis très heureuse et fière d'avoir travaillé avec toi.*

Je présente mes plus chaleureux remerciements à tous les membres de l'équipe Explore du LIRMM, tous membres de l'UMR ITAP, tous les membres du projet Adap2E et tous mes collègues de l'IUT GEII de Montpellier.

Je tiens également à remercier tous les membres de l'UFR SEGMI, département mathématiques et informatique pour l'ambiance exceptionnelle qui règne dans le laboratoire.

Je tiens à remercier tous ceux qui ont participé à la réalisation de ce travail.

Dédicaces

Du profond de mon coeur, je dédie ce travail de thèse à tous ceux qui me sont chers :

A la mémoire de ma chère mère Raja SAFFAR

Je ne saurais exprimer mon grand chagrin en ton absence. Aucun mot ne pourrait décrire mes sentiments.

J'aurais aimé que tu sois à mes côtés ce jour là, et entendre tes cris de joie "zagared" dans toute la salle.

J'aurais aimé que tu voies votre petite fille ayant obtenue le plus haut diplôme en France et que tu me dises combien tu es fière de moi.

J'aurais aimé te remercier devant tout le monde pour tous les sacrifices que tu as consentis pour mon éducation et que je partage avec toi l'un des plus beaux moments de ma vie.

Que ton âme se repose en paix.

A mon cher père Rachid SADDEM

Source inépuisable de tendresse, d'amour, de patience et de sacrifice. Ta prière et ta Bénédiction m'ont été d'un grand secours tout au long de ma vie pleine de bonheur.

Chaque ligne, chaque mot et chaque lettre de cette thèse t'exprime l'infinité de ma reconnaissance, mon grand respect et mon amour sans limite.

Merci pour ta grande confiance, ton soutien et tous tes encouragements. J'espère que tu es fier de moi.

J'implore Dieu le tout puissant de te préserver et t'accorder bonne santé et longue vie.

A mes chères soeurs Ramla, Rabeb, Rahma et mon cher frère Rayen

Des soeurs et un frère comme on ne peut trouver nul part ailleurs.

Je ne saurai traduire sur du papier l'affectation et l'amour que j'éprouve à vos égards.

Je vous remercie pour votre soutien, votre confiance et vos encouragements.

Puisse Dieu vous protéger, garder et renforcer notre fraternité.

Je vous souhaite tout le bonheur du monde.

A mon cher mari Djamel Edine YAGOUBI

De tous les êtres humains, tu es le meilleur.

Mon amour éternel, la source de ma joie et de mon bonheur, nulle dédicace ne pourrait exprimer mes sentiments et mon profond attachement à toi.

Je remercie le bon dieu qui a croisé nos chemins.

Puisse le bon dieu nous procurer santé et longue vie.

A mes beaux-parents et leur enfants

Vous m'avez accueilli les bras ouverts.

Je vous dédie ce travail en témoignage de ma profonde affectation et mon grand amour.

Que Dieu vous protège et vous procure bonne santé et bonheur.

A mes proches et tous mes amis

Trouvez ici l'expression de mon profond respect et mon fidèle attachement.

Table des matières

Résumé

L'agriculture de précision (AP) s'est fortement développée au cours des dernières décennies avec les progrès des technologies de localisation, des capteurs et des techniques de télédétection. Le principe de l'AP est de rechercher et mettre en oeuvre la Bonne action au Bon moment et au Bon endroit ("3B"), avec l'objectif d'améliorer l'efficience de l'agriculture suivant les critères d'une agriculture durable. Nous considérons dans cette thèse l'évaluation de la faisabilité d'opérations agricoles vis à vis de critères relevant de l'AP, en faisant appel à des techniques de modélisation et de vérification formelles.

Pour modéliser et vérifier ces opérations, une représentation des dynamiques temporelles et spatiales est nécessaire. Le model-checking de systèmes d'automates temporisés avec des requêtes dans une logique temporelle arborescente répond à ces besoins, les positions spatiales pouvant être représentées de façon ad-hoc dans le cadre de ces formalismes. Trois exemples d'opérations agricoles sont considérés dans ce mémoire. La première est relative au calcul d'une séquence optimale de commandes pour une pulvérisation de précision en viticulture. La seconde concerne la récolte sélective en viticulture. La dernière est relative à la vérification d'une mission de robotique agricole. Nous étudions dans ces exemples l'atteignabilité d'un état cible pour l'opération, ou l'atteignabilité avec un critère de coût optimal.

Pour pallier au problème d'explosion combinatoire rencontré dans les cas traités, une méthodologie de décomposition pour le model-checking d'atteignabilité a été développée. Les résultats expérimentaux avec et sans décomposition sont présentés pour les 3 exemples d'opération étudiés. La technique de décomposition est appliquée sur 2 des 3 exemples et les résultats expérimentaux montrent son efficacité.

Mots clés : systèmes à événements discrets, synthèse de contrôleur, décomposition, model checking, agriculture de précision, optimisation.

Abstract

Precision agriculture (PA) has grown strongly in recent decades with advances in locating methods and sensor and remote sensing technologies. The principle of the PA is to seek and implement the Right action at the Right time and in the Right place ("3R"), with the objective of improving the efficiency of agriculture according

to the criteria of agriculture sustainable. We consider in this thesis the evaluation of the feasibility of agricultural operations with respect to the criteria set in a PA recommendation, using formal modeling and verification techniques.

To model and verify these operations, a representation of temporal and spatial dynamics is necessary. The model-checking of timed automata systems with requests in temporal tree logic answers these needs. The spatial positions can be represented in an ad-hoc manner within the framework of these formalisms. Three examples of agricultural operations are considered in this thesis. The first relates to the computing of an optimal sequence of commands for a precision spraying in viticulture. The second concerns selective harvesting in viticulture. The last one is related to the verification of an agricultural robotics mission. We study in these examples the reachability of a target state for the operation, or the reachability with an optimal cost criterion.

To overcome the combinatorial explosion problem encountered in the treated cases, a decomposition methodology for reachability model-checking has been developed. The experimental results with and without decomposition are presented for the 3 examples of operation studied. The decomposition methodology was applied to 2 of the 3 examples and the experimental results indicate its efficient.

Introduction générale

1 Évolutions technologiques et précision de l'agriculture

Après une période durant laquelle la mécanisation et l'automatisation ont fortement progressé en agriculture, les technologies relevant du numérique pour une agriculture intensive en information s'imposent désormais pour contribuer à l'efficience des procédés de production agricole. Ces dernières décennies ont vu l'apparition de nombreuses technologies innovantes, notamment de nouveaux capteurs, de nouveaux robots, des nouvelles techniques de localisation, de télédétection, etc. L'usage de ces technologies se répand. Le défi actuel de l'agriculture, dans un contexte de croissance de la population mondiale, est d'améliorer la qualité et la quantité des produits tout en respectant l'homme et l'environnement.

Une des voies pour y parvenir est d'améliorer la précision des opérations agricoles. En effet, l'agronomie a jusqu'à des périodes récentes privilégié le principe d'homogénéité des parcelles pour deux raisons. La première est que la plupart des parcelles étaient, en Europe, de taille suffisamment réduite pour justifier l'uniformité des interventions sur chaque parcelle ou ensemble de parcelles. La deuxième tenait à l'absence des moyens techniques et économiques nécessaires pour qualifier la variabilité intra-parcellaire. Les exploitations agricoles sont cependant aujourd'hui de plus en plus grandes. Cela a conduit, et conduit encore, à une réorganisation et un regroupement des parcelles, notamment pour les grandes cultures. Rester dans une approche conventionnelle de la gestion des cultures peut alors conduire à mettre en oeuvre des opérations culturales trop uniformes [41].

La variabilité intra-parcellaire est à l'origine du développement de l'agriculture de précision. Dans les années 80, aux USA, des cartes avec des analyses tous les 100 mètres sur la fertilité des sols, ont mis en évidence la nécessité d'une modulation des apports d'engrais suivant les zones [127]. La gestion de la variabilité intra-parcellaire pour la fertilisation a été une des premières applications de la notion d'agriculture de précision pour les céréales [41]. La notion d'agriculture de précision s'applique également à d'autres pratiques culturales. Par exemple, dans le cas de la pulvérisation phytosanitaire, l'uniformité de traitement peut provoquer un sous ou un sur-traitement des parcelles, ou de zones des parcelles, avec pour résultat une

efficience moins grande des produits épandus [115]. Or de nombreux produits phytosanitaires présentent des effets négatifs pour l'environnement et la santé humaine [101]. Une meilleure efficience des traitements, favorisant ainsi une réduction des quantités épandues, pourrait contribuer à la réduction de ces risques. Si on considère la culture de la vigne, à l'échelle du couvert végétal, pulvériser les parcelles de façon uniforme peut conduire à des doses de produit rapportées à la surface de feuillage très variables [48], avec les surdosages et sous-dosages associés, et également générer des pertes hors du végétal cible.

Le développement et le déploiement des technologies de capteurs (localisation GPS, capteurs fixes ou embarqués, télédétection, imagerie aérienne et embarquée sur drones,...) incitent les agronomes et les informaticiens à concevoir des méthodes et des outils pour évaluer avec précision la variabilité intra-parcellaire. Par exemple, l'outil Farmstar [76] permet de moduler la dose d'azote à partir d'images satellites. Il y a une tendance à utiliser de plus en plus les approches d'agriculture de précision. Ceci se justifie, d'une part, par l' effort technologique qui continue à se développer et d'autre part par le travail de sensibilisation auprès des agriculteurs.

Par ailleurs, l'agriculture mécanisée qui était largement déployée dans les pays industrialisés dans les années 70 nécessite la disponibilité d'une main d'oeuvre qualifiée pour effectuer des tâches et traitements localisés, répétitifs et pénibles. Ces travaux agricoles sont complexes, mais aussi dangereux (exposition à des produits phytosanitaires, accidents de travail comme résultant, par exemple, du renversement d'une machine agricole). Ils nécessitent donc un coût important en terme de temps, d'effort physique, de présence sur le terrain.

C'est pourquoi l'objectif de la robotique agricole est de réduire la pénibilité des travaux agricoles et d'augmenter la productivité. La robotique agricole a connu un "âge d'or" dans la première moitié des années 80 [65]. En France, les premiers pas des robots en agriculture ont été menés par le Cemagref (Antony [1], Montpellier et Rennes) et les laboratoires bordelais LARFRA et ENSAM. A l'époque, plusieurs robots ont été conçus, et certains concepts ont été un vrai succès. Ainsi, la moitié des exploitations laitières sont aujourd'hui équipées d'un robot de traite des vaches. D'autres robots n'ont pas abouti à des systèmes commercialisés. Cela a été le cas, par exemple, pour le robot cueilleur de pommes Magali [18]. Dans les années 90, les raisons de ces difficultés tenaient à des limitations technologiques, à leurs coûts de mise en oeuvre, et aux cadences productives nécessaires pour en justifier l'emploi. Aujourd'hui, les technologies sont en évolution constante, les capteurs sont de plus en plus précis et de moins en moins chers, les ordinateurs sont de plus en plus performants. Ceci conduit à rendre l'automatisation et la robotisation de plus en plus attractive. Le coût de développement des robots et leur prix de vente restent encore une difficulté pour le secteur agricole mais de nombreuses entreprises innovent

1. Le centre Antony se trouve à la région Île-de-France

dans ce domaine comme par exemple les entreprises françaises Naïo technologies [2] et VitiBot [3], et à l'international, les entreprises japonaises OPTim [4] et Yanmar [5], les entreprises américaines Harvest CROO [6], Agribotix [7], etc, et enfin en australie, l'entreprise australienne swarmfarm [8] et le Centre australien de robotique de terrain (Université de Sydney) [9]. Elles développent toutes des logiciels et des robots qui ont un bon niveau d'autonomie dans l'exécution de leurs tâches.

Nous venons d'évoquer la tendance de l'agriculture à devenir de plus en précise par l'emploi de technologies innovantes. En tant que telle, la gestion d'une culture peut-être considérée comme un procédé dont on cherche à vérifier les qualités ou améliorer l'efficience. C'est ce que nous abordons dans la section suivante.

2 Vers la vérification des procédés agricoles

Le procédé cultural conduit, avec un certain nombre d'opérations culturales tout au long de la saison (préparation du sol, semis, fertilisations, pulvérisations pour la protection de la culture, récolte, etc), et selon les conditions (de climat, de sol, etc), à des effets pouvant être appréciés au plan quantitatif (quantité récoltée, intrants consommés, etc). L'opération culturale elle-même peut-être vue comme un procédé, ayant une dynamique temporelle et spatiale, et pouvant impliquer un ou plusieurs équipements agricoles. En agriculture de précision comme en robotique agricole, on s'attend à ce que l'opération culturale respecte une préconisation spatialisée et soit conforme à des exigences :

- Précision spatiale de l'action effectuée,
- Précision quantitative de l'action effectuée (dose de fertilisant ou de produit phytosanitaire par exemple),
- Durée du chantier de mise en œuvre de l'opération culturale compatible avec des contraintes météorologiques, économiques et sociales,
- Consommation d'intrants et d'énergie conforme aux prévisions,
- etc

2. https://www.naio-technologies.com/
3. http://www.vitibot.fr/nos-robots/
4. https://en.optim.co.jp/news-detail/11510
5. https://www.yanmar.com/global/technology/robotics.html
6. http://www.harvestcroorobotics.com/
7. https://agribotix.com/
8. https://www.swarmfarm.com/
9. https://sydney.edu.au/engineering/our-research/robotics-and-intelligent-systems/australian-centre-for-field-robotics/agriculture-and-the-environment.html

A la conception d'un équipement, ou lors de la définition d'une nouvelle méthode culturale utilisant des équipements existants, il est donc pertinent de vérifier que le procédé mis en œuvre respecte ces exigences. Avant les tests sur le terrain, cette vérification pourrait être initiée en recourant à la simulation du procédé. Mais celle-ci a cependant l'inconvénient de n'être pas exhaustive : tous les comportements potentiels ne sont pas obligatoirement évalués. Pour cette raison, le respect des exigences n'est pas garanti. Dans cette thèse, nous proposons de mettre en œuvre des techniques de modélisation formelle des systèmes d'opération agricole et d'évaluation formelle des exigences, pour aider à la conception d'opérations culturales et des équipements qui réalisent ces opérations. Les techniques qui sont proposées ici relèvent de la vérification formelle par model-checking. D'après la littérature, ces techniques et outils ont peu été employés pour des questions agricoles et environnementales ([57], [86]). Nous proposons donc ici des applications originales du model-checking à l'agriculture de précision et à la robotique.

3 Contributions et plan de la thèse

Le principe général du model-checking est d'évaluer l'ensemble des comportements possibles vis-à-vis du respect d'exigences traduites sous forme de propriétés. Ces exigences portent notamment sur la capacité d'un engin agricole à évoluer et agir avec la précision souhaitée dans l'environnement agricole.

Or, l'environnement agricole est très complexe. En fonction de la tâche agricole ou de déplacement de l'engin agricole, et du milieu traversé, la manière de représenter l'environnement agricole peut être différente. Pour des raisons de simplification, nous distinguerons deux types d'environnements :

- Environnement ouvert où l'engin agricole peut se déplacer librement et changer à tout moment de direction. L'environnement et les déplacements peuvent alors être modélisés, par exemple, par une grille à deux dimensions ("2D") représentant les positions de l'engin.

- Environnement structuré pour une culture plantée en rangs. Le mouvement de l'engin doit alors impérativement longer ces alignements.

  - Lorsque seul le déplacement dans l'inter-rang est considéré, l'engin avance dans une seule direction, la représentation spatiale est mono-dimensionnelle ("1D"). La granularité de la représentation dans cet espace peut être assez fine, suivant la précision souhaitée dans l'action agricole et les contraintes sur les systèmes physiques.
  - Lorsque l'engin se déplace dans une parcelle formée par des rangs, outre le parcours dans l'inter-rang, on considère le choix des rangs à parcourir et leur succession, ainsi que le sens choisi pour parcourir le rang. On parlera ici de "1D+".

La manière de modéliser l'environnement, l'engin et la tâche a donc un impact direct sur la complexité des modèles. La représentation spatiale du système évalué aura donc un impact sur la combinatoire des propriétés analysées par model-checking.

Dans ce mémoire, nous traitons des problèmes 1D, 1D+ et 2D. Les modèles que nous avons développés sont fondés sur la théorie des automates temporisés et des automates temporisés de coût.

Les trois problématiques agricoles abordées dans ce mémoire relèvent chacune d'une des dimensions spatiales que nous venons d'évoquer. La première de ces trois problématiques est relative à la pulvérisation de précision. Nous avons développé une méthode qui permet de calculer une séquence de commande optimale pour la pulvérisation dans un rang à partir des données lidar. Il s'agit d'une problématique en environnement 1D. La deuxième est relative à la récolte sélective. Il s'agit de calculer une logistique de récolte optimale à partir d'une carte de préconisation spatiale qui distingue des zones de qualité, avec deux qualités de raisins. Les zones sont identifiées indépendemment des rangs et donc chaque rang peut avoir des raisins des deux qualités. Il s'agit d'une problématique en environnement 1D+. Ces deux problématiques concernent les cultures pérennes structurées en rangs, comme la vigne. La troisième problématique est relative à un environnement non structuré. Il s'agit de vérifier, a priori, qu'une mission robotique, formulée comme une propriété d'atteignabilité particulière, peut être réalisée. Il s'agit d'une question/problématique en environnement 2D. Dans tous ces cas, la vérification de propriétés d'atteignabilité conduit à un problème d'explosion combinatoire. Pour y faire face, une méthode de décomposition a alors été proposée.

Le présent manuscrit est structuré autour de quatre parties. La première partie présente un état de l'art sur l'agriculture de précision, et sur les méthodes formelles de model-checking. Elle positionne aussi les travaux de la thèse dans le cadre du projet ANR jeune chercheur Adap2e [90]. Nos travaux de thèse s'inscrivent en effet dans le contexte applicatif et environnemental de ce projet de robotique et contribuent à ses résultats méthodologiques en matière de planification.

La deuxième partie présente via les 3 problématiques, évoquées plus haut, des méthodologies d'application des techniques du model-checking en agriculture de précision. Ces méthodes sont confrontées au problème de l'explosion combinatoire, problématique couramment rencontrée en model-checking, qui tient en particulier ici à la représentation des déplacements dans l'espace. La troisième partie présente la solution que nous avons proposée pour résoudre ce problème d'explosion combinatoire sous la forme d'une méthodologie de décomposition. Cette approche est mise en œuvre sur deux des trois problématiques présentées en partie 2. La quatrième partie conclut et donne des perspectives à ces travaux de thèse.

Première partie

Contexte et problématique

1 Agriculture de Précision & Robotique

Sommaire

1.1 Introduction

L'agriculture assure l'essentiel des besoins alimentaires d'une population humaine en constante augmentation. Par ailleurs, dans des pays comme la France, le nombre des exploitations agricoles diminue, et la surface travaillée par exploitant augmente. La tendance est à l'augmentation de la productivité horaire de la main-d'oeuvre [54]. A ces besoins de productivité auxquelles répondent les technologies agricoles depuis plusieurs décennies s'ajoutent des préoccupations d'efficience qui concernent également l'environnement et la santé. On observe aussi une demande pour une alimentation de qualité. Si cette qualité favorise le revenu des agriculteurs, elle pourrait contribuer à la soutenabilité de l'agriculture.

L'agriculture de précision (AP) est susceptible de contribuer à satisfaire ces différentes demandes vis-à-vis de l'agriculture. En effet, l'AP peut être vue comme une gestion spécifique des cultures qui permet d'adapter les actions agricoles à la situation locale sur le terrain afin que la bonne action soit effectuée au bon endroit et au bon moment [95] [1]. En particulier, si on considère les intrants agricoles, comme la

1. Extrait du texte : *One generic definition could be "that kind of agriculture that increases the number of (correct) decisions per unit area of land per unit time with associated net benefits". This moves the focus a little away from simply spatial resolution to one involving the fineness of decisions in both space and time.*

fertilisation, l'eau, les pesticides, l'objectif de l'AP est d'en favoriser une utilisation parcimonieuse.

La robotique agricole peut également contribuer à l'objectif d'une agriculture durable en reconsidérant les choix de mécanisation. En effet, les besoins de productivité horaire ont conduit jusqu'ici à des équipements de grande taille et à des procédés qui favorisent le débit et la régularité plutôt que la gestion de la variabilité. La perspective pour la robotique agricole est notamment de réduire la taille et le coût des équipements qui doivent acquérir une autonomie partielle dans leur fonctionnement pour pouvoir adresser cette variabilité.

Dans ce chapitre, nous allons expliquer les fondamentaux de l'agriculture de précision, puis nous situerons dans cette perspective la démarche de modélisation et de vérification à la base de nos travaux.

1.2 Historique et concepts de l'agriculture de précision

L'AP est un sujet de recherche depuis de nombreuses années. Plusieurs congrès internationaux sont organisés autour de cette notion. L'ECPA, "European Conference on Precision Agriculture", a lieu tous les deux ans depuis 1997. La société savante ISPA ("International Society on Precision Agriculture") organise l'ICPA "International Conference on Precision Agriculture" tous les deux ans depuis 1996. On trouve également l'"Asian Australasian Conference on precision Agriculture" (depuis 2005, tous les deux ans), l'"International Conference on Agricultural Technology and Precision Agriculture" (depuis 1997, chaque année), etc. Il y a aussi des conférences dédiées à l'application des nouvelles technologies à l'agriculture, dont l'agriculture de précision, comme la conférence EFITA organisée par l'"European Federation for Information Technology in Agriculture, Food and the Environment". Le journal scientifique international de référence pour l'AP est le «Journal of Precision Agriculture», apparu en 1999.

Si l'AP intéresse de nombreux scientifiques de différentes communautés à l'international, il faut noter qu'il n'existe pas de définition unique de l'AP [95]. Les définitions de l'AP peuvent être classées en 2 catégories [91] : des définitions à orientation technologique qui mettent l'accent sur le progrès technique et les outils (développement des capteurs de rendement, des capteurs GPS *Global Positionning System* ....) et des définitions à orientation agronomique agricole qui mettent l'accent sur la finalité de la démarche. On peut par exemple se référer à la thèse de Jerémy Lherbier intitulée "Valorisation de l'information géographique en agriculture de précision" [91] qui présente plusieurs définitions et les compare. Nous retiendrons deux définitions complémentaires.

1. La Chambre des représentants des États-Unis en 1997 a donné une première définition de l'AP : "Un système qui vise à augmenter l'efficacité, la productivité et la rentabilité à long terme de la production agricole et de la production tout entière tout en minimisant les impacts non désirés sur la faune et l'environnement".

2. Le laboratoire d'agriculture de précision de l'université de Sydney définit la gestion des cultures spécifiques au site (*SSCM, site specific crop management*)) : "Une forme d'AP où les décisions sur l'application des ressources et les pratiques agronomiques sont améliorées pour mieux correspondre aux exigences du sol et des cultures, car elles varient sur le terrain".

La première définition de l'AP est une définition générale qui englobe toutes les productions agricoles qu'elles soient animales ou végétales. Elle donne les finalités de l'agriculture de précision mais elle ne précise pas comment on atteint ces objectifs. La deuxième définition précise les moyens à employer, et notamment l'exploitation de la variabilité spatiale de la culture.

Le concept d'AP est en fait né au USA au début des années 80 sous le nom de "specific farming". Les travaux menés à cette époque concernent la fertilité des sols. Des cartes avec des analyses tous les 100 mètres ont mis en évidence la nécessité d'une modulation des apports d'engrais suivant les zones. Plus précisément, des études des taux de Phosphore (P) et de Potassium (K) ont montré une grande variabilité de la fertilité des parcelles [127]. Certains affirment que la variabilité intraparcellaire est l'origine du développement de l'agriculture de précision. En Europe, l'AP est apparue pour la première fois en Angleterre à la fin des années 80 pour les cultures herbacées (cultures de céréales, cultures maraîchères, cultures des espèces de la famille des légumineuses). Un point d'histoire sur l'agriculture de précision est disponible dans [49]. La pratique de l'agriculture de précision en France a débuté à la fin des années 90 [127].

Le début de l'AP a été confronté à plusieurs types de verrous : socio-économiques, technologiques et agronomiques. Une description détaillée de ces verrous est disponible dans [107]. A l'époque, les verrous socio-économiques identifiés concernaient principalement les coûts liés à l'AP, l'attitude de l'exploitant et sa résistance au changement due à son niveau de formation considéré comme souvent limité. Les verrous technologiques étaient essentiellement liés à la conception des machines agricoles (problèmes d'entretien, machines mal adaptées aux petites parcelles, etc), et aux capteurs (inexistants ou trop coûteux). Les verrous agronomiques identifiés concernaient le manque de conseillers connaissant ces techniques et le manque de données détaillées, notamment sur les variations intraparcellaires de la qualité des sols.

C'est grâce au développement de nouveaux des capteurs et à la réduction des coûts, notamment des coûts des informations GPS, que l'agriculture de précision s'est développée rapidement. Un récepteur GPS permet de se localiser pratiquement n'importe où en extérieur, dès lors que suffisamment de satellites sont visibles du

récepteur, et n'importe quand. Ce système a été utilisé au départ en agriculture pour mesurer la surface des terrains (arpentage) et pour la cartographie de rendements [49]. En ce qui concerne les outils utilisant la technologie GPS, aujourd'hui largement disponibles pour les agriculteurs en grandes cultures, des experts techniques comme Caroline Desbourdes [2] les classent en 2 catégories : applications du GPS pour la précision en agriculture et applications du GPS pour l'agriculture de précision.

Dans le premier cas, le GPS est utilisé pour repérer avec précision la position d'un appareil agricole dans un champ afin d'effectuer des actions dont la localisation est certaine et précise. Un premier exemple est le guidage ou l'autoguidage. L'autoguidage consiste à guider un appareil en suivant une trajectoire de référence sans intervention du chauffeur. Ceci permet de diminuer, par exemple, la surface de recouvrement entre deux passages, afin d'optimiser le débit de chantier et le tassement des sols.

Dans la seconde catégorie d'applications, l'AP consiste à réaliser les bonnes actions au bon endroit, en fonction des caractéristiques du sol ou de la culture en place. C'est la modulation intraparcellaire, notion qui recouvre *la gestion des cultures spécifiques au site* vue plus haut. Toujours d'après Caroline Desbourdes [53], la modulation intraparcellaire comporte 3 étapes :

- Étape 1 : Caractériser la variabilité liée au milieu physique et au végétal.
- Étape 2 : Établir une carte de préconisation pour chaque point de la parcelle.
- Étape 3 : Réaliser la modulation dans le champ (modification de dose par exemple) de manière manuelle ou automatique.

Avec la maturité des technologies dans certains secteurs de l'agriculture et la disponibilité grandissante de données, les scientifiques académiques du domaine de l'AP ont mis en avant les questions méthodologiques, au-delà de l'évaluation du potentiel de nouvelles technologies pour répondre aux objectifs de l'AP. C'est pourquoi la section suivante présente une schématisation des grandes fonctions de l'agriculture de précision, qui va nous permettre de situer les contributions de ce mémoire.

1.3 Les fonctions de l'agriculture de précision

Comme défini dans [118, 96] et illustré dans la figure 1.1, l'agriculture de précision peut être vue comme une stratégie de gestion spatialisée et d'amélioration cyclique des procédés culturaux basée sur 5 fonctions :

- L'*observation* : elle consiste à l'acquisition des données sur le végétal. Certaines données sont coûteuses à acquérir et sont disponibles avec une faible résolution

2. spécialiste en agriculture de précision chez Arvalis, Journée Agriculteurs du 08/12/15 à St Jean de Linières (49) - Arvalis, diapos disponibles sur www.evenements-arvalis.fr, voir aussi [53]

spatiale. D'autres au contraire, grâce à des technologies embarquées sur les équipements ou la télédétection, sont acquises avec une résolution importante.

- La *caractérisation* : elle permet de transformer les données obtenues lors de la phase d'observation en information agronomique.
- La *préconisation* : elle permet de produire un conseil technique explicite en se basant sur les informations agronomiques élaborées lors de la caractérisation.
- L'*application* : c'est la mise en oeuvre de la préconisation, le conseil agronomique est appliqué sur la ou les parcelle(s) par l'opérateur.
- Le *référencement spatial* : c'est la fonction centrale qui permet de spatialiser les données récoltées, les informations agronomiques et l'application de la préconisation.

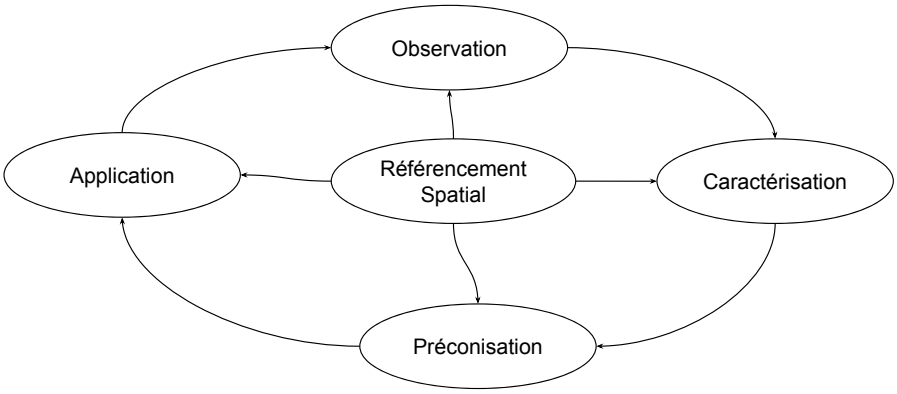


FIGURE 1.1 – Les fonctions de l'agriculture de précision

L'objectif de cette thèse est d'évaluer comment les méthodes formelles, notamment le model-checking, et leurs outils associés, peuvent contribuer à ces fonctions.

McBratney et ses co-auteurs insistent sur la nécessité de bien prendre en compte la dimension temporelle [95]. Celle-ci peut par exemple intervenir dans la mise en œuvre des préconisations. En effet, la résolution spatiale de la préconisation doit être compatible avec les possibilités techniques des équipements, et notamment la précision de travail ainsi que les temps de réponse des actionneurs. Dans le cas où les caractéristiques des équipements sont limitantes, la préconisation doit être adaptée pour pouvoir être réalisée.

Pour prendre en compte le fonctionnement des équipements, Roudier et ses co-auteurs [109] ont proposé la notion d'index d'opportunité technique (TOi : *Technical Opportunity index*). Ce dernier détermine la manière dont la préconisation doit être mise en œuvre sur la parcelle en considérant les caractéristiques des équipements. En

effet, la mise en œuvre d'un traitement modulé à l'échelle intraparcellaire ne peut se faire qu'en respectant les contraintes associées au matériel utilisé. Il convient donc de proposer des méthodes permettant, sur la base d'informations spatialisées, de savoir si une parcelle est apte ou non à recevoir un traitement modulé en fonction des caractéristiques des machines qui seront utilisées ou des opérations qui seront effectuées. Le TOi permet de classer les parcelles par rapport à l'intérêt de procéder à un traitement modulé intraparcellaire.

Nos travaux présentés dans la partie II propose des approches permettant la mise en oeuvre de certaines fonctions de l'Agriculture de Précision et permettant la prise en compte du fonctionnement des équipements.

Au chapitre 4, nous décrivons une méthode permettant de calculer une séquence de commande optimale pour la pulvérisation à partir de données LiDAR. La méthode couvre les 3 fonctions caractérisation, préconisation et application, et met en œuvre le model-checking sur les fonctions préconisation et application. Dans le cas où la préconisation spatialisée conduit à une mise en œuvre complexe du point de vue logistique, par exemple pour organiser des trajets optimisés entre parcelles ou au sein de la parcelle, les techniques de model-checking combinées à de l'optimisation peuvent contribuer à une phase "application" efficiente.

Au chapitre 5, nous proposerons une méthode pour la récolte sélective. Son principe est de disposer d'une carte préconisant un tri suivant la qualité attendue sur la récolte, et de calculer une logistique de récolte optimale respectant cette préconisation.

Enfin, dans l'hypothèse où un robot autonome ou partiellement autonome conduirait des tâches d'agriculture de précision dans un ensemble de parcelles séparées entre elles et de la ferme par des espaces partiellement structurés, il conviendrait de vérifier, au moment du calcul de la mission du robot, si les conditions sont réunies pour que la mission soit réalisable. Nous verrons aux chapitres 3 et 6 que les techniques de model-checking, et notamment la vérification de l'atteignabilité d'un état souhaité, peuvent contribuer à cette vérification.

1.4 Limites de la simulation pour l'agriculture de précision

En général, en agriculture de précision, la validation des systèmes s'effectue par des tests sur le terrain [52] et par simulation [126, 125]. En langage courant, il y a une confusion entre validation, vérification et simulation. Rappelons tout d'abord la définition de la validation. La validation est un processus d'évaluation d'un système lorsque le développement de ce dernier est terminé. Elle consiste à s'assurer que le résultat attendu est bien atteint et elle permet de répondre à la question "Est ce que le système fait bien ce qu'il est censé faire ?".

La vérification est un processus qui vise à s'assurer que les choses sont faites conformément à ce qui avait été défini dans les spécifications et elle permet de répondre à la question "Est ce que le système a été construit correctement (tel que nous l'avions spécifié) ?". Parmi les techniques de vérification, on trouve la vérification formelle qui fait partie des méthodes formelles. L'intérêt de procéder à une étape de vérification dans le cadre du processus plus global de la validation est de pouvoir bien évaluer, en cas de problème rencontré lors de la validation, si ce problème a pour origine les spécifications, qui ne seraient pas compatibles avec les objectifs, ou si le problème réside dans la non-conformité du système aux spécifications.

La simulation consiste à explorer un sous-ensemble des cas possibles et le système est alors validé seulement par rapport à ce qui a pu être simulé. La vérification formelle a l'avantage par rapport à la simulation que tous les cas possibles sont explorés. Dans la suite, nous présentons les limites de la simulation pour l'agriculture de précision.

En raison de sa flexibilité et de son efficacité, la simulation est la méthode la plus utilisée pour valider vis-à-vis des spécifications et des objectifs notamment la phase de mise en œuvre de la préconisation (fonction application dans la figure 1.1) en agriculture de précision. Le principe est d'utiliser un outil permettant l'exécution d'un modèle du système dans un contexte typique de son utilisation future. L'observation du comportement de ce système et sa comparaison avec celui attendu permet alors de détecter d'éventuelles erreurs de conception ou des performances insuffisantes par rapport au cahier des charges. Par exemple, l'approche du TOi [109] permet une mise en oeuvre de la simulation en calculant les erreurs sur l'action réalisée par rapport à une carte de préconisation de référence. Considérons, par exemple, deux cartes d'une parcelle formée de 5 rangs : la première est une carte de préconisation et la deuxième est une carte des résultats. La figure 1.2 présente ces deux cartes.

Rang 1	TA	**TB**	TA	TA
Rang 2	TB	TB	TB	**TA**
Rang 3	TA	TA	TA	TA
Rang 4	TB	TB	**TA**	TB
Rang 5	TB	TB	TB	**TA**

Carte de préconisation

Rang 1	TA	TA	TA	TA
Rang 2	**TA**	**TA**	TB	TB
Rang 3	**TB**	TA	TA	TA
Rang 4	**TA**	TB	TB	TB
Rang 5	TB	TB	TB	TB

Carte des résultats

FIGURE 1.2 – Exemple de cartes de préconisation et de résultats

L'approche TOi permet de calculer les erreurs de la carte des résultats par rapport à la carte de préconisation. Sur chaque point de la parcelle, la carte de préconisation définit une consigne à appliquer qui peut être soit d'appliquer le traitement A

(TA) soit d'appliquer le traitement B (TB). Une simplification de cette carte consiste à appliquer TA sur les rangs 1 et 3 et TB sur les rangs 2, 4 et 5. Cette simplification génère des erreurs (en gras soulignés dans la carte de gauche de la figure 1.2) par rapport à la carte de préconisation initialement définie. Au moment d'application de cette carte simplifiée, il est possible d'avoir des retards d'application de la consigne. Ces retards augmentent les erreurs. Par exemple, au début du rang 2 dans la carte des résultats, TA a été appliquée au lieu de TB. Les cases en bleu représentent les erreurs par rapport à la préconisation simplifiée.

Pour les systèmes automatiques en agriculture, la conception du modèle de simulation s'appuie en général sur une relation déterministe entre l'état du champ et l'action d'une machine agricole. Plus concrètement, le modèle doit comprendre un état initial du champ, un modèle de perception de l'état du champ par la machine, un modèle d'action de la machine sur l'état du champ, et une loi de commande qui lie la perception et l'action de la machine [126].

Cependant, dans ce cadre classique, il est difficile de considérer des évènements potentiellement non contrôlables comme par exemple l'apparition d'une panne. Mais il peut y avoir d'autres événements non contrôlés, imparfaitement contrôlés ou pour lesquels le(s) moment(s) d'occurrence ne sont pas connus ou prédictibles : erreur aléatoire sur la mesure, survenue d'une personne ou d'un animal devant la machine, début d'une pluie, état de la culture imprévu (verse [3] dans un champ par exemple), mauvais état de terrain, glissement, etc.

Lorsqu'il est possible de déterminer des modèles stochastiques décrivant la survenue des événements non contrôlés, alors un modèle de simulation initialement déterministe peut être étendu à une approche de simulation stochastique. Mais cette tâche peut être difficile voire inapplicable en pratique du fait du coût nécessaire pour caractériser au plan statistique les phénomènes à simuler.

Par ailleurs, l'élaboration d'une loi de commande déterministe n'est pas nécessairement la meilleure solution pour obtenir un résultat optimisé. Pour y parvenir, il faut alors paramétrer la loi de commande et rechercher par simulation-optimisation les meilleurs valeurs pour ces paramètres. Ces valeurs peuvent dépendre de la situation rencontrée. Une approche plus adaptée peut être de concevoir un contrôle optimal. Dans ce cas, la spécification de la commande peut-être partielle, c'est-à-dire non simulable de façon déterministe si l'optimisation n'est pas incluse dans la simulation.

Pour toutes ces raisons, à savoir la prise en compte d'événements non contrôlés et l'intérêt que l'on peut avoir à spécifier la commande de façon incomplète pour laisser des marges d'optimisation, l'emploi de la simulation pour valider une méthode d'agriculture de précision peut trouver ses limites. Il ne sera en effet pas possible de

3. En agriculture, la verse est un accident de végétation touchant certaines cultures, elle entraîne le plus souvent une baisse importante du rendement, voire la perte de la récolte, elle peut être due à une forte pluie ou à un vent violent ou même aussi à des attaques parasitaires, etc

simuler tous les cas possibles et il sera donc impossible de garantir que le modèle fonctionne dans toutes les configurations pouvant être rencontrées.

L'emploi d'autres méthodes de validation, telle que les méthodes formelles, s'avère donc a priori pertinent. La vérification de modèles diffère de la simulation en ce sens que toutes les possibilités pour un modèle et la propriété à vérifier sont explorées de manière exhaustive.

Nous pouvons illustrer notre objectif d'employer des techniques de vérification formelle en considérant l'exemple suivant, énoncé pour un pulvérisateur dans un champ de vigne. Un objectif de pulvérisation de précision peut être de pulvériser un rang avec une quantité globale minimale de produit tout en garantissant une protection suffisante pour chaque plant de vigne afin d'éviter l'initiation de maladies. Supposons que nous disposions d'un pulvérisateur automatique qui permettent de choisir entre plusieurs configurations de pulvérisation. Considérons que le rang est formé par n blocs de végétation et que plusieurs configurations sont possibles pour chaque bloc. Supposons encore qu'une instrumentation d'agriculture de précision et une bonne connaissance des effets des configurations de pulvérisation permette de déterminer pour chaque bloc deux commandes qui garantissent une protection suffisante. En d'autres termes, dans le cadre d'une approche d'agriculture de précision, nous pouvons construire une carte de préconisation spatialisée du champ, avec, pour chaque bloc, deux commandes envisageables.

Par simulation déterministe, il n'est possible d'appliquer qu'une seule configuration pour un bloc donné. A chaque endroit de la carte, il faut alors choisir a priori la commande locale à appliquer, simuler cette commande, et évaluer la qualité du résultat global. Si on souhaitait trouver la meilleure solution par simulation, alors il faudrait effectuer un très grand nombre de simulations en changeant, pour chaque instant, la commande choisie parmi toutes les commandes possibles. Il faut donc trouver une autre méthode, plus flexible, permettant de tirer parti de la préconisation flexible (deux possibilités de configuration pour chaque bloc). Une voie est d'utiliser les techniques de vérification formelle telles que le model-checking.

Dans ce contexte, le model checking permet de modéliser l'ensemble de l'opération de pulvérisation et la dynamique de l'équipement utilisé. A partir de ce modèle, il est alors possible de vérifier des propriétés qui permettent d'explorer toutes les possibilités pertinentes. L'exploration exhaustive permet, d'une part, de vérifier que la préconisation flexible est applicable et d'autre part de calculer la séquence de commandes optimale à appliquer sur le rang. L'exhaustivité que nous permet d'adresser les méthodes de vérification formelle est un paradigme le plus souvent inaccessible par le biais de la simulation.

Dans le chapitre suivant, les bases du model-checking seront présentées.

2 Vérification formelle par model-checking

Sommaire

2.1 Introduction

Dans le chapitre précédent, nous avons présenté un état de l'art sur l'agriculture de précision. Nous avons positionné l'ensemble de nos travaux par rapport aux fonctions de l'agriculture de précision. Nous avons également souligné la nécessité d'utiliser des méthodes de vérification formelle pour l'agriculture de précision.

Nous allons maintenant présenter un état de l'art sur les méthodes, formalismes et outils de vérification formelle.

Les méthodes de vérification formelle sont utiles à la vérification des Systèmes Temps Réel (STR). Ces derniers sont des systèmes informatiques qui ont des contraintes de temps du fait de l'interaction avec leur environnement, avec d'autres systèmes et le cas échéant avec des opérateurs. Ils sont présents dans de nombreux secteurs d'activités : l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites), l'industrie de production, l'automobile (systèmes ferroviaires ou véhicules autonomes), etc. Dans certains secteurs, les STR sont des systèmes critiques car ils contrôlent des fonctions critiques sur le plan de la sécurité. Il est alors nécessaire de vérifier ces systèmes de façon formelle pour garantir la fiabilité de leurs modes de fonctionnement.

Il existe plusieurs méthodes formelles de vérification. Ces méthodes visent à assurer que le système fonctionne correctement. Elles peuvent être classées en deux grandes catégories [97] : les méthodes syntaxiques et les méthodes sémantiques.

Les méthodes syntaxiques sont des preuves mathématiques assistées par ordinateur ou automatiques, souvent appelées preuve automatique de théorèmes. Étant donnée une description mathématique d'un système, un ensemble d'axiomes et un ensemble de règles d'inférences, la preuve automatique de théorèmes consiste à laisser l'ordinateur prouver les propriétés désirées. La preuve automatique de théorèmes peut être très efficace. Elle a l'avantage, par rapport aux méthodes sémantiques, d'être indépendante de la taille de l'espace d'états, et peut donc s'appliquer sur des modèles avec un très grand nombre d'états, ou même sur des modèles dont le nombre d'états n'est pas déterminé. Cependant, elle possède deux grandes limitations : d'une part elle nécessite une grande expertise dans la modélisation du système et des règles d'inférence, ensuite tous les systèmes ne sont pas modélisables sous la forme d'axiomes mathématiques.

Les méthodes sémantiques se basent sur l'application de la sémantique d'un modèle du système (son exécution) pour vérifier si les propriétés sont satisfaites. Elles sont applicables uniquement si le modèle a un nombre fini d'états. L'approche la plus populaire est le model-checking ([2], [32]) qui s'appuie sur deux types de formalismes : un formalisme pour représenter les comportements du système et

un formalisme pour représenter les propriétés à vérifier. Le premier, de type machine à états, modélise le système par des états et des événements entre ces états (synchronisation, évolution interne, évènements liés à une entrée du système, etc). On peut évoquer parmi ces formalismes les plus utilisés pour la modélisation des STR : les automates temporisés ([4],[3]) et les réseaux de Petri temporels ([63]). Le second formalisme nécessaire pour exprimer les propriétés est la logique temporelle. Plusieurs logiques temporelles utilisées pour le Model-Checking (MC) seront présentées dans ce chapitre dans la section 2.4. Dans tous les cas, la propriété devant être vérifiée est exprimée comme une formule de logique pouvant être vérifiée ou non (`true, false`) sur un état donné du système. Plus précisément, le principe du model-checking consiste à vérifier l'ensemble des propriétés censé refléter le bon comportement d'un système sur l'ensemble des états accessibles et des chemins d'exécution du modèle de ce système. L'exhaustivité de la vérification garantie le bon fonctionnement du système dans tous les cas auxquels il pourra être exposé si, bien évidemment, le modèle du système est correct. Cependant, cette énumération exhaustive peut poser des problèmes en temps de traitement et surtout, en taille de l'espace mémoire nécessaire pour stocker les états et réaliser la vérification. C'est le phénomène d'explosion combinatoire.

Le model-checking est la méthode que nous avons choisie pour répondre à notre problème de modélisation et vérification des systèmes en agriculture de précision. Nous allons dans ce chapitre en expliquer les bases, point de départ pour la compréhension de tous nos travaux. Le reste du chapitre est organisé comme suit : dans la section suivante, nous donnons une définition détaillée du model-checking et nous décrivons les différentes phases du processus qui permet de le mettre en œuvre. Nous employons dans les travaux présentés dans ce mémoire le formalisme des automates temporisés, nous le présentons en section 2.3. La section 2.4 propose une introduction aux logiques temporelles. La section 2.5 décrit la vérification de propriété. La section 2.6 donne un aperçu des outils de MC. Enfin, la section 2.7 décrit la problématique d'analyse d'accessibilité à coût optimal. Cette section se termine par une description de l'outil *memtime*[1] [19] qui permet de mesurer le temps et la mémoire consommée au moment de la vérification.

2.2 Processus de Model-Checking

Au début des années 1980, Joseph Sifakis et Jean-Pierre Queille en France et, de manière indépendante, Edmund M. Clarke et E. Allen Emerson aux Etats-Unis ont proposé une méthode automatique de vérification formelle des systèmes : Le model-checking ([103],[45]). Cela leur a valu le prix Turing en 2007. Le model-checking ([2],[9]) permet d'assurer que le modèle d'un système est conforme à ses spécifications. Celles-ci sont décrites sous forme de propriétés logiques temporelles. Les deux

1. memtime est disponible à l'adresse `http://www.update.uu.se/~johanb/memtime`

champs d'applications classiques des techniques de model-checking sont la validation des composants électroniques et la validation des protocoles de communication [9, 46, 55]. Les techniques de model-checking sont de plus utilisées dans des applications industrielles ([44], [43]). Un ouvrage proposant une synthèse historique a été édité en 2008 ([66]).

Le processus de model-checking se base sur 3 phases :

1. La première est la phase de modélisation formelle du comportement du système via des machines à états. Elle sera décrite dans la section 2.3.

2. La deuxième est la phase de description de la spécification attendue du système via des propriétés, exprimées en langages logiques. Cette phase sera décrite dans la section 2.4.

3. La troisième est la phase de vérification. Elle consiste à déterminer si le modèle satisfait la propriété. Si la propriété n'est pas satisfaite alors un contre-exemple peut être fourni. Cette phase sera décrite dans la section 2.5.

Nous allons maintenant décrire chacune de ses phases.

2.3 Modélisation formelle

La modélisation formelle est la première phase du processus de vérification par model-checking. Elle consiste à modéliser le comportement du système via des machines à états. Cette phase est particulièrement délicate car elle nécessite un sens aigu de modélisation et d'abstraction. Généralement, le système est décrit en langage naturel, ce qui nécessite une traduction en langage formel de cette description. Cela peut induire des erreurs ou nécessiter des hypothèses de la part du modélisateur si la description en langage naturel comporte des ambiguïtés. Le niveau d'abstraction choisi est un élément critique du processus de modélisation, car il doit refléter correctement le fonctionnement du système tout en garantissant la bonne compréhension du modélisateur mais aussi la faisabilité et la justesse du processus de vérification. Il est également possible d'avoir plusieurs niveaux d'abstraction d'un même système, en fonction de l'utilité du modèle et des propriétés à vérifier. Dans cette partie du processus de modélisation formelle, l'expertise du modélisateur sera déterminante pour construire un modèle à la fois correct et adapté au model-checking.

Dans la section suivante, nous présenterons les différents formalismes de modélisation.

2.3.1 Les différents formalismes pour le model-checking

Le model-checking se base sur une modélisation formelle du système à base de machines à états. Il fait donc appel à des formalismes qui représentent des états et des transitions entre états.

Nous avons besoin pour appliquer les techniques de model-checking à l'agriculture de précision et à la robotique agricole de représenter plusieurs phénomènes :

- Changement d'état instantané : par exemple, réception d'un message (si le temps de réception est négligeable devant les autres phénomènes).
- Tâche ayant une durée non nulle fixe : par exemple, récolte d'un rang de vigne à vitesse connue.
- Tâche ayant une durée non nulle imparfaitement connue ou qui dépend de l'interaction : par exemple, le parcours à pied d'un rang de vigne par un viticulteur.

Les transitions correspondent à des événements. Les tâches, quand-à-elles, sont représentées comme des états. Elle peuvent être des abstractions de phénomènes continus. Par exemple, l'ouverture d'une porte pourrait se représenter par une succession de 3 états, *"porte ouverte"*, *"porte en cours d'ouverture"* et *"porte fermée"*, mais on peut aussi ne s'intéresser qu'aux états *"porte ouverte"* et *"porte fermée"*.

La dimension temporelle du comportement du système doit pouvoir être traduite dans le modèle de ce dernier. Pour répondre à ces besoins en modélisation, nous avons besoin de systèmes dits temporisés. Les formalismes les plus utilisés pour la modélisation des systèmes temporisés, dans le cadre d'un processus de vérification formelle, sont les automates temporisés (AT) ([4],[3]) et les réseaux de Petri temporels (RPT) ([63]).

Les réseaux de Petri temporels et les automates temporisés peuvent modéliser les mêmes phénomènes à savoir les systèmes dynamiques à états discrets et temporisés [22]. C'est pourquoi, il existe même des méthodes permettant une traduction structurelle des réseaux de Petri temporels vers les automates temporisés [40]. Le choix d'un formalisme peut également dépendre des outils logiciels disponibles. Pour ces travaux, nous avons fait le choix d'utiliser les automates temporisés. Ce formalisme est en effet très puissant, il est adapté à nos besoins et il permet l'expression de tous les éléments nécessaires pour représenter notre système. De plus, l'outil UppAal que nous utilisons dans le cadre de ce travail et que nous présenterons en section 2.6.2, est un logiciel de validation et vérification formelle extrêmement performant qui s'appuie sur un formalisme des AT que nous allons maintenant détailler.

2.3.2 Les automates temporisés

Les Automates Temporisés (AT), tels que définis dans [4], sont un formalisme simple dans sa syntaxe mais dont la puissance d'expressivité permet de modéliser le comportement temporisé des systèmes temps réel. Ce formalisme rajoute aux automates finis un ensemble fini de variables à valeur réelle positive ou nulle, appelées horloges. Des contraintes sur les horloges peuvent être définies pour décrire un invariant temporel associé à un état (cet invariant doit être vrai tant que le système

est dans cet état) ou exprimer une garde temporelle sur une transition (conditionnant le tir de cette transition). Ce type de représentation, largement utilisé pour la spécification de l'automatisation industrielle (avionique, Intel, IBM, Motorola, etc) [47], a également été développé dans la recherche pour l'agriculture ou la gestion des écosystèmes ([70], [86], [57]).

Donnons une définition formelle d'un Automate Temporisé [20] : soit C un ensemble d'horloges et $B(C)$ l'ensemble des formules qui sont des conjonctions de contraintes atomiques sur ces horloges de la forme $x \bowtie n$ et $x - y \bowtie m$ pour $x, y \in C$, $\bowtie \in \{<, \leq, =, \geq, >\}$, m et n des entiers. Un automate temporisé est un quadruplet (L, E, l_0, inv) où L est un ensemble d'états de contrôle, $l_0 \in L$ est l'état initial, $E \subseteq L \times \sum \times B(C) \times 2^C \times L$ est l'ensemble des transitions entre états avec un état source, une "action" (événement) dans un ensemble fini d'actions $\sum$, une garde (contraintes sur les horloges), l'ensemble d'horloge à initialiser, et un état destination. $inv : L \rightarrow B(C)$ est une application, appelée invariant, qui associe à chaque état de contrôle une conjonction de contraintes d'horloges de la forme x $\bowtie$ c où $\bowtie in \{<,\leq\}$.

La figure 2.1 permet de donner un exemple d'un tel automate. Cet automate a deux états $S0$ et $S1$. L'état initial est l'état $S0$. L'état $S0$ a un invariant x<= 10, l'automate peut donc rester dans cet état tant que l'horloge x est inférieure ou égal 10. Lorsque la valeur de x est supérieure à 7, la transition vers l'état $S1$ peut avoir lieu (action a). Cette transition ré-initialise la valeur de l'horloge x à 0.

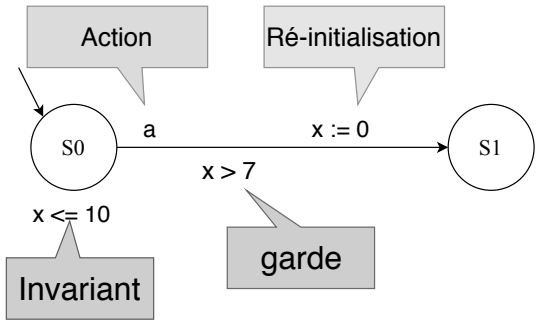


FIGURE 2.1 – Exemple d'un automate temporisé

Dans la suite, nous présentons les différentes types de propriétés, les connecteurs logiques et les différents types de logiques.

2.4 Spécifications formelles pour la vérification

Cette section présente une introduction aux logiques temporelles. Elle commence par une classification des propriétés temporelles puis présente les connecteurs logiques et les différents types de logiques.

2.4.1 Types de propriétés

Dans la littérature sur la vérification, plusieurs types de propriétés temporelles sont identifiées. Nous présentons dans ce paragraphe une classification de ces propriétés souvent rencontrée dans les travaux de spécification des systèmes temps réel. Elle distingue les 6 catégories de propriétés suivantes :

- *Propriétés de sûreté* : Une propriété de sûreté énonce que quelque chose de non souhaitable ou dangereux ne se produira jamais.
- *Propriétés de vivacité* : Une propriété de vivacité énonce, que sous certaines conditions, quelque chose de souhaitable finira par avoir lieu.
- *Propriétés d'atteignabilité* : Une propriété d'atteignabilité, également appelée propriété d'accessibilité, énonce qu'un état respectant cette propriété peut être atteint ou non.
- *Propriétés d'invariance* : Une propriété d'invariance énonce que tous les états d'un système respecte une propriété.
- *Propriétés d'équité* (ou de vivacité générale) : Cette propriété énonce que, sous certaines conditions, quelque chose aura lieu (ou n'aura pas lieu) un nombre infini de fois.
- *Propriétés d'absence de blocage* : Cette propriété énonce que le système ne se bloquera jamais, c'est-à-dire qu'il n'existe pas d'état bloquant. Un état bloquant correspond à un état duquel le système ne peut pas évoluer vers un état final.

La propriété d'atteignabilité est utilisée dans de nombreux types d'applications, à la fois dans le domaine de la recherche et dans l'industrie. La thèse de Sebastiaan J. van Schaik intitulée *Answering reachability queries on large directed graphs* présente plusieurs applications de la propriété d'atteignabilité [121]. On peut citer par exemple :

- Analyse de code source [7] : La propriété d'atteignabilité permet de chercher du code mort (des méthodes qui ne peuvent et ne seront jamais atteintes à partir de la méthode principale d'une application).
- Analyse des réseaux sociaux [121] : Les réseaux sociaux peuvent être modélisés comme des graphes : les sommets représentent les personnes et les arcs représentent les connexions entre les personnes. Les informations d'atteignabilité peuvent être utilisées pour déterminer par transitivité si une personne est un ami d'un ami de $\cdots$ d'un ami. Cela permet au réseau de poser des restrictions sur le fait qu'une personne puisse ou non voir la page de profil d'une autre personne.

- Planification d'itinéraire : A première vue, il ne semble en général pas très intéressant du point de vue pratique de savoir si un chemin existe ou non pour atteindre un endroit donné. La réponse est souvent positive, dès lors que le point à atteindre est bien relié au réseau routier. Par contre si on contraint la question d'accessibilité (par exemple, ne pas utiliser une route à péage, durée du voyage,...), le problème devient plus complexe. On peut également rechercher le chemin le plus court ou le plus rapide respectant le cas échéant des contraintes. Nous verrons que cette fonction peut être adressée à l'aide d'une forme spécifique de model-checking abordée en section 2.7.3.

- Analyse des propriétés de graphe : D'une façon plus générale, la propriété d'atteignabilité permet de vérifier des caractéristiques sur les graphes comme par exemple, la connexité.

Toutes ces propriétés sont exprimables en utilisant le formalisme de la logique temporelle et ses connecteurs logiques.

2.4.2 Les connecteurs logiques

Il existe trois types de connecteurs :

- Les connecteurs propositionnels ($\wedge$, $\vee$, $\neg$,$\Rightarrow$,$\Leftrightarrow$)

- Les connecteurs temporels : (X, F, G, U)

- Les quantificateurs de chemins : (A, E)

Les connecteurs propositionnels permettent d'établir une liaison entre les différentes propriétés atomiques. $\wedge$ désigne la conjonction, $\vee$ désigne la disjonction, $\neg$ désigne la négation, $\Rightarrow$ désigne l'implication et $\Leftrightarrow$ désigne l'équivalence.

Les connecteurs temporels ou quantificateurs d'états permettent d'exprimer l'enchaînement d'évènements au long d'un chemin sans introduire le temps explicitement.

Soit p, q deux propriétés atomiques.

- Xp (neXt) désigne l'*état suivant*. Dans l'état suivant p est vraie.

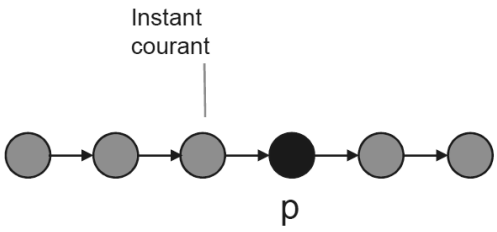


FIGURE 2.2 – Connecteur X

- Fp (Future) désigne *inévitablement*. A partir de l'instant courant, il y aura obligatoirement un état futur où p sera vraie.

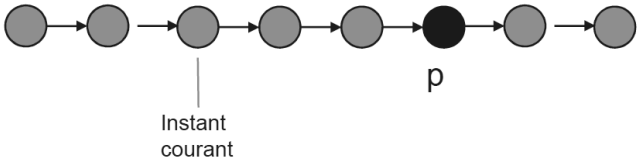


FIGURE 2.3 – Connecteur F

- Gp (Globally) désigne *toujours*. A partir de l'instant courant et dans tous états suivants, p est et restera vraie.

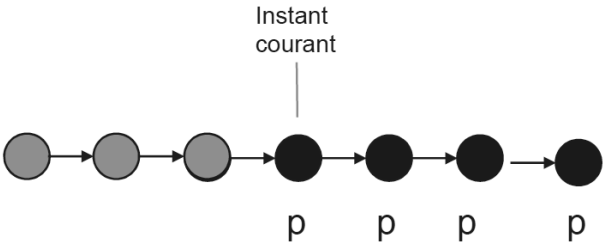


FIGURE 2.4 – Connecteur G

- U (Until) désigne *jusqu'à*. pUq signifie que p est toujours vraie jusqu'à un état où q est vrai.

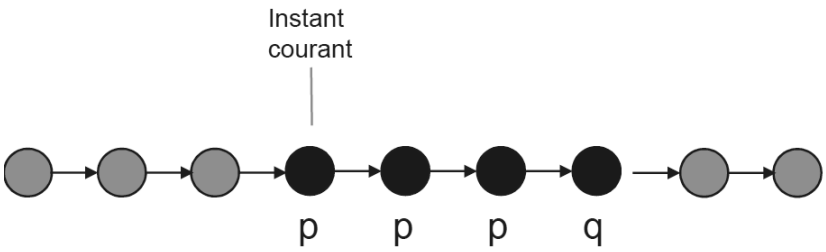


FIGURE 2.5 – Connecteur U

Les quantificateurs de chemins permettent d'exprimer des propriétés en logiques arborescentes :

- Ap désigne *pour tous les chemins*. La propriété p est vérifiée sur tous les chemins.

- Ep désigne qu'*il existe au moins un chemin*. Il existe, au moins, un chemin où p est vrai.

Dans les logiques arborescentes, les combinaisons d'opérateurs les plus courantes et les plus utiles sont les 4 suivantes (voir Figure 2.6) :

- AGp : p sera toujours vraie, sur tous les chemins.

- EGp : p sera vraie tout au long d'au moins un chemin.

- AFp : p sera vraie au moins une fois sur tous les chemins.

- EFp : il existe au moins un chemin où p sera vraie au moins une fois.

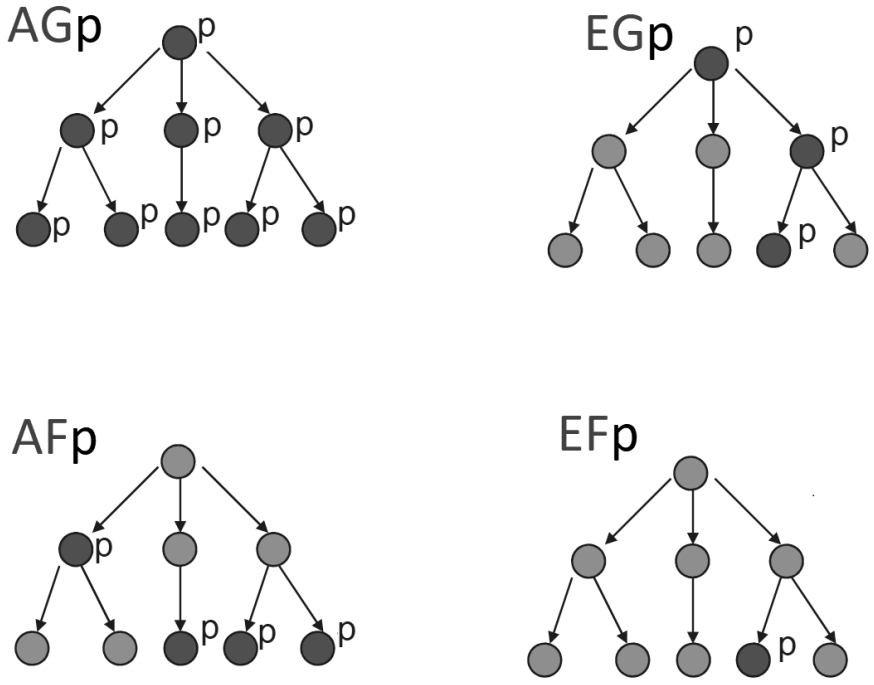


FIGURE 2.6 – Combinaison de connecteurs pour les logiques arborescentes

Dans la suite, nous présentons les différents types de logiques temporelles.

2.4.3 Types de logiques temporelles

Les logiques temporelles permettent d'exprimer les différentes propriétés temporelles. Pour ces logiques, lorsqu'elles ne sont pas temporisées, le temps est une donnée abstraite permettant de représenter la succession des évènements (ordre d'occurrence) mais qui n'est pas quantifiée. Par contre, les logiques temporelles temporisées permettent de manipuler explicitement une quantification du temps.

Dans la littérature, on peut distinguer quatre familles principales de logiques temporelles : LTL (Linear Temporal Logic) [102], CTL (Computation Tree Logic)[45], CTL*[58] et TCTL (timed CTL)[2]. La logique temporelle linéaire (LTL) a d'abord été proposée par Amir Pnueli en 1977 [102], dans le but de vérifier des programmes informatiques. Quatre ans plus tard, en 1981, Edmund M. Clarke et Allen Emerson inventent CTL et le model-checking [45]. CTL* fut définie par Emerson et Joseph Y. Halpern en 1986 [58]. Ces familles sont classées en deux grandes catégories : arborescentes et linéaires [82]. Ces catégories sont distinguées par la nature de la relation d'ordre considéré pour l'ordonnancement temporel entre les états :

- Si l'ordre est total, le comportement du système est vu comme une séquence d'exécution. Le (déroulement du) temps est linéaire. A chaque instant, il n'y a qu'un seul futur possible. La spécification se fait via les logiques temporelles linéaires (LTL).

- Si l'ordre est partiel, le comportement du système est vu comme un arbre décrivant l'ensemble des exécutions possibles. Le temps est arborescent. A chaque instant, plusieurs futurs sont possibles, selon l'action qui sera effectuée. La spécification se fait via les logiques temporelles arborescentes (CTL, TCTL, CTL*).

Nous allons maintenant présenter rapidement chacune de ces familles de logiques temporelles.

La logique LTL
La logique temporelle LTL est une logique linéaire. Cette logique ne permet pas de différencier les chemins. La vérification d'une formule LTL sur un système de nature arborescente revient à utiliser un quantificateur A (de façon implicite).

Soit AP un ensemble fini de propriétés atomiques. L'ensemble des formules LTL est défini par la grammaire abstraite suivante [111] :

$$p, q ::= p \cup q \mid Xp \mid p \wedge q \mid \neg p \mid P_1 \mid P_2 \mid ...$$

avec $AP = \{P_1, P_2\}$ un ensemble de propositions atomiques. Les autres connecteurs booléens $\top, \bot, p \vee q, p \Rightarrow q$ et $p \Leftrightarrow q$ sont définis via les abréviations usuelles. Les modalités LTL classiques F et G sont obtenues par les équivalences suivantes :

$$Fp \equiv \top \cup p \quad et \quad Gp \equiv \neg F \neg p$$

La logique CTL*
La logique CTL* est une extension de LTL. C'est une logique arborescente (*branching-time logic*) très expressive. Elle permet une utilisation libre des quantificateurs de chemins E et A et elle ne présente aucune restriction sur tous les connecteurs logiques. L'utilisation de CTL* directement est assez rare, elle est souvent déclinée en "sous-logique", donc la plus connue est CTL, et TCTL en version temporisée. L'ensemble des formules CTL* est défini par la grammaire abstraite suivante [111] :

$$p, q ::= Ep \mid p \cup q \mid Xp \mid p \wedge q \mid \neg p \mid P_1 \mid P_2 \mid ...$$

E est le quantificateur pour l'existence d'un chemin, et le quantificateur universel de chemin A est défini par équivalence :

$$Ap \equiv \neg E \neg p$$

La logique CTL
La logique CTL est une logique arborescente, restriction de CTL*. Elle permet une utilisation libre des quantificateurs de chemins E et A, mais les connecteurs temporels X, F, G, U doivent être précédés directement de A ou E. Ainsi la logique CTL peut être définie avec quatre modalités : $E\cup, A\cup, EX$ et AX. Sa syntaxe est la suivante [111] :

$$p, q ::= E(p \cup q) \mid A(p \cup q) \mid EXp \mid AXp \mid p \wedge q \mid \neg p \mid P_1 \mid P_2 \mid ...$$

Les modalités EF, AF, EG et AG énoncées plus haut peuvent être définies par des équivalences.

La logique TCTL
La logique TCTL (Timed CTL) est une extension de la logique CTL. Elle ajoute une notion de temps quantitatif qui permet de vérifier des propriétés incluant des durées. Les formules TCTL sont construites d'après la syntaxe suivante [30] :

$$p, q ::= P_1 \mid \neg p \mid p \vee q \mid Ep \cup_{Iq} \mid Ap \cup_{Iq})$$

où I désigne un intervalle temporel.

Un exemple typique de propriété TCTL peut être une propriété de temps de réponse borné telle que :

$$AG(a \rightarrow AF \leq_{56} b)$$

Cette propriété permet de vérifier qu'à chaque fois que l'évènement a se produit, l'évènement b doit forcément se produire dans les 56 unités temporelles suivantes.

Cette logique, comme la plupart des extensions des logiques temporelles (temporisées, paramétrées, probabilistes), entraîne une complexité supplémentaire dans son utilisation, en particulier dans le processus de vérification par model-checking. Nous ne l'utiliserons donc pas dans nos travaux.

2.4.4 Expression des propriétés pour le model-checking

Expression des propriétés en CTL

La logique CTL est la logique la plus adaptée à la vérification de propriété sur des structures arborescentes. Or le model-checking est basé sur la génération de l'espace d'états d'un modèle sous la forme d'un graphe des états possibles. L'utilisation de CTL permet des vérifications fines grâce à l'opérateur E. En effet, la vérification de propriétés LTL sur une structure arborescente correspond à l'utilisation implicite de l'opérateur A appliquée à la formule LTL désirée, ce qui ne permet pas de différencier finement les différents chemins [82].

Les propriétés temporelles classiques s'écrivent en CTL comme suit :

- *Propriété de sûreté* : Elle s'écrit sous la forme : $AGNONp$ avec p une propriété qui exprime un problème ou un danger.
- *Propriété de vivacité* : Elle s'écrit sous la forme : AFp.
- *Propriété d'atteignabilité* : Elle s'écrit sous la forme : EFp.
- *Propriété d'invariance* : Elle s'écrit sous la forme : AGp.
- *Propriété d'équité* : CTL n'exprime pas directement l'équité.
- *Propriété d'absence de deadlock* : Elle s'écrit sous la forme : $AG(EXtrue)$.

Focus sur la propriété d'atteignabilité

La propriété qui nous intéresse le plus dans cette thèse est la propriété d'atteignabilité. Si on considère le model-checking du point de vue du problème de l'explosion combinatoire, la vérification d'une propriété d'atteignabilité est un problème plus facile à résoudre que, notamment, la vérification d'une propriété de sûreté. En effet, la propriété peut être dite satisfaite dès qu'un état vérifiant la propriété désirée a été trouvé dans un chemin quelconque partant de l'état initial considéré. Alors qu'une propriété de sûreté nécessite le parcours de l'ensemble des états (et donc des chemins) du système. Pourtant, la question de l'atteignabilité a fait l'objet de nombreux travaux de recherche au cours des deux dernières décennies ([116], [5], [121]). En effet, les ressources du système informatique de vérification sont limitées. Vérifier une propriété en réduisant le temps nécessaire pour la vérification et en optimisant les ressources computationnelles nécessaires est donc une nécessité qui motive les travaux de recherche. Dans ce mémoire, nous ne chercherons pas seulement à vérifier des propriétés mais aussi à optimiser en termes de temps et d'utilisation de la mémoire le processus de vérification.

2.5 La vérification de propriété

La troisième phase du processus du MC est la phase de vérification. Elle consiste à déterminer si le modèle satisfait la propriété étudiée. Si celle-ci n'est pas satisfaite alors un contre-exemple peut être fourni. L'analyse de ce dernier peut aider à déterminer la source du problème ayant engendré la violation de la propriété. Deux cas sont possibles : 1) le problème provient du processus de modélisation : une erreur dans le modèle du système ou dans celui de la propriété. Le modélisateur devra alors corriger cette erreur. 2) ou bien le problème provient du système lui-même : en l'état des spécifications, la garantie du bon fonctionnement du système est donc impossible. Il s'agira alors d'interagir avec le concepteur initial afin de réfléchir aux solutions de corrections à apporter.

2.5.1 Algorithme de model-checking

Cette section expose, illustré sur un exemple simple, les principes de base du MC de propriétés exprimées en logique CTL sur un modèle exprimé en AT.

2.5.1.1 Construction du graphe d'états

Nous allons considérer comme exemple une version très simplifiée du problème de récolte qui sera exposé dans le chapitre 5. Soit un robot qui récolte une parcelle formée de seulement 2 rangs. Chaque rang i a deux extrémités (Ni et Si). Un rang ne devant être récolté qu'une seule fois, chaque extrémité d'un rang ne sera donc visitée qu'une seule fois. Si le robot se déplace à une extrémité du rang, il faut qu'il récolte ce rang. Par exemple, si le robot se déplace en S1, il doit obligatoirement effectuer le trajet S1N1. La figure 2.7 permet de représenter la parcelle ainsi que les coûts des chemins (ici, cela représente la distance à parcourir).

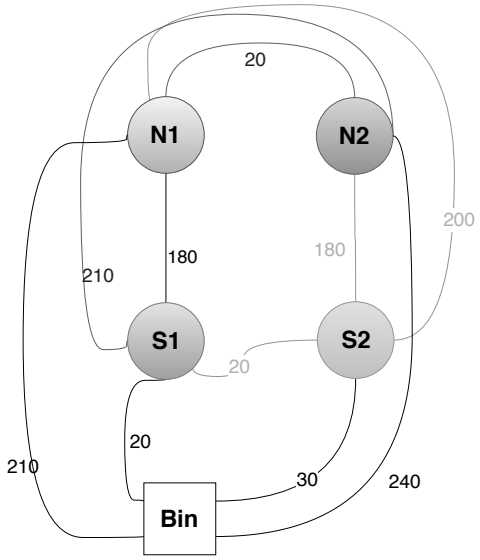


FIGURE 2.7 – Représentation de la parcelle

A l'état initial, le robot est situé à un endroit précis de la parcelle appelé *bin* (la benne). Après la récolte de chaque rang, le robot a le choix entre récolter un autre rang ou aller à l'emplacement *bin* pour vider la récolte. L'objectif est de récolter tous les rangs, éventuellement en allant vider la récolte si besoin, puis retourner à l'emplacement initial *bin*.

Pour une raison de simplification, l'automate détaillé qui permet de décrire cet exemple n'est pas fourni. Cependant, il a globalement la structure de la figure 2.7 mais il représente, en plus, les différentes contraintes du système, en particulier les

contraintes de visites. Par exemple, si le robot se déplace de N1 à S1, il n'a plus le droit de revenir à S1.

Le graphe d'états de cet exemple est représenté par la figure 2.8. Il représente tous les chemins possibles du robot dans la parcelle. Les états en couleur rouge sont les états qui vérifient l'objectif final de récolte.

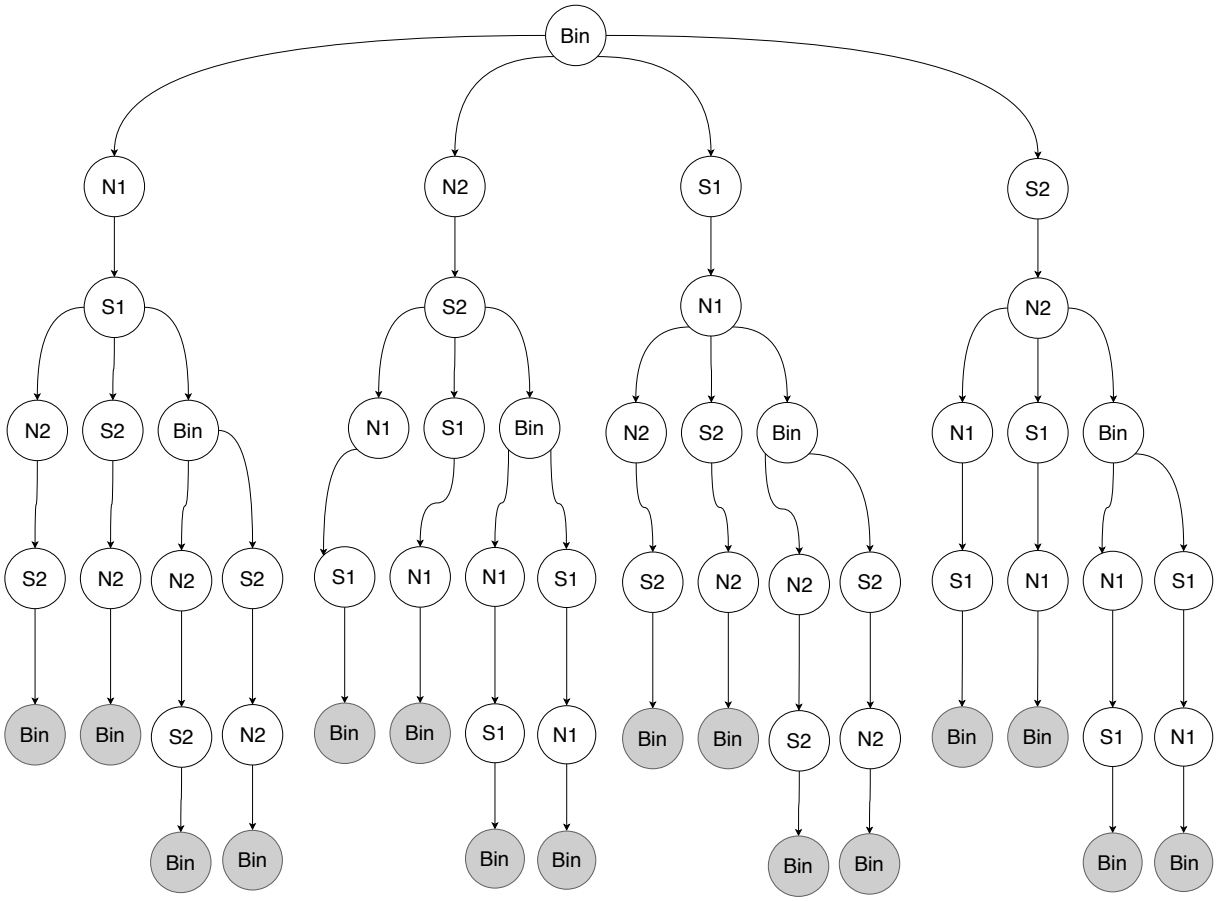


FIGURE 2.8 – Graphe d'état généré pour la récolte de la parcelle de la Figure 2.7

Après avoir construit le graphe d'état de l'exemple, nous allons maintenant le parcourir à la recherche des chemins menant aux états finaux. Plusieurs stratégies d'exploration de graphe existent, la section suivante en présente quelques unes.

2.5.1.2 Parcours du graphe

Dans cette section, nous nous intéressons au parcours du graphe, en nous appuyant sur l'exemple précédent. Il peut y avoir un parcours en largeur d'abord ou en profondeur d'abord [117]. Ce sont deux options différentes de parcours du graphe d'états dans le cas d'un parcours partiel, c'est-à-dire lorsque que le parcours (ou la construction) du graphe d'états s'arrête dès qu'une propriété donnée est vérifiée.

Cela s'appelle le model-checking à la volée (*on-the-fly*). La figure 2.9 représente l'espace d'état généré pour la vérification de la propriété avec un parcours en largeur d'abord et la figure 2.10 représente l'espace d'état généré pour la vérification de la propriété avec un parcours en profondeur d'abord. Les noeuds en couleur bleu sont les noeuds déjà explorés au moment où la propriété (l'objectif final de récolte) est vérifiée pour la 1ère fois. Le parcours en largeur consiste, à partir du premier sommet du graphe, à visiter tous les sommets successeurs. On répète l'opération tant qu'aucun sommet n'a encore vérifié la propriété. Dans notre exemple, le parcours en largeur d'abord nécessite l'exploration de presque tout l'espace d'états.

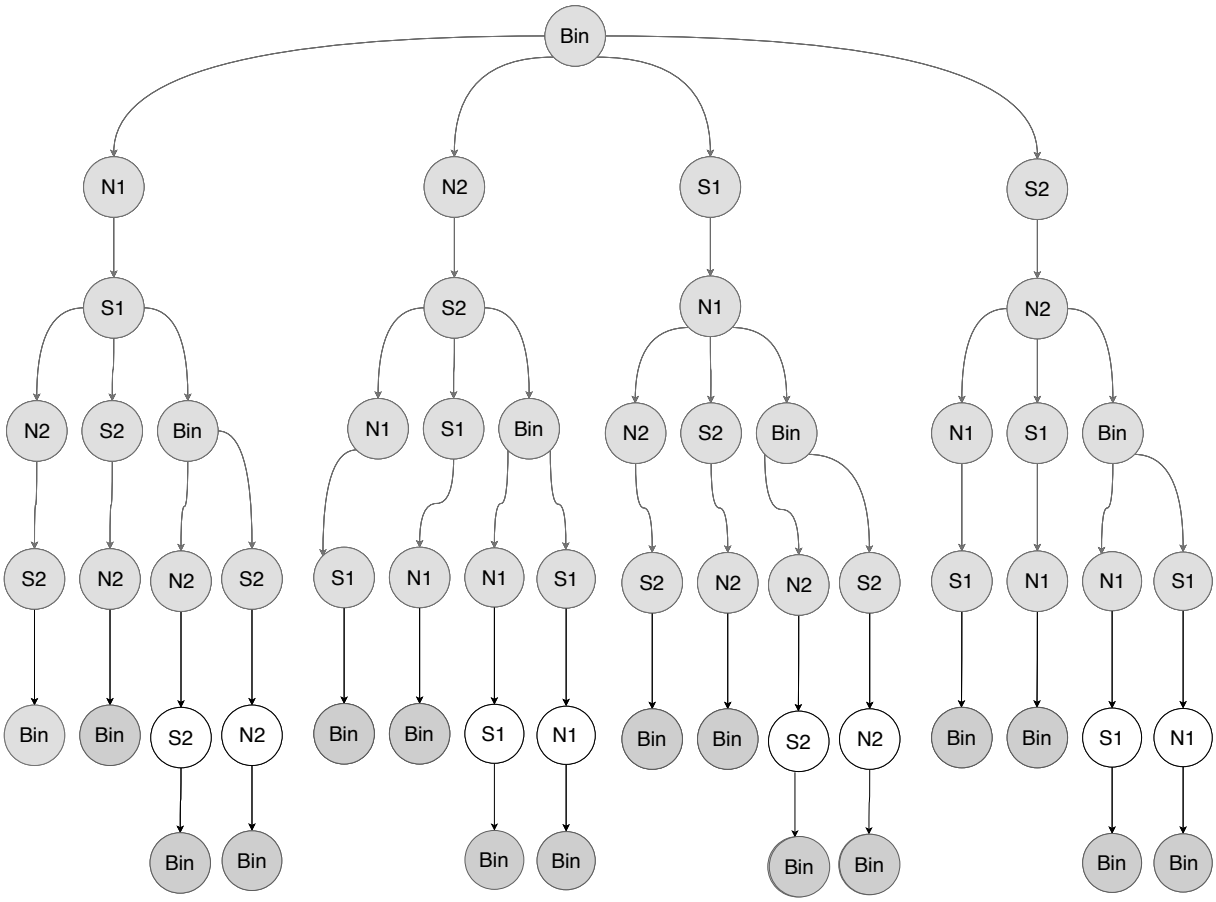


FIGURE 2.9 – Illustration du parcours en largeur d'un graphe d'état

Le parcours en profondeur d'abord explore un sommet, puis un de ces successeurs, et essaie d'aller le plus loin possible dans les successeurs. Quand ce n'est pas possible, il revient en arrière et essaie de nouveau de parcourir en profondeur à partir d'un second successeur. Dans notre cas, l'exploration en profondeur permet de ne parcourir qu'une seule branche de l'espace d'états.

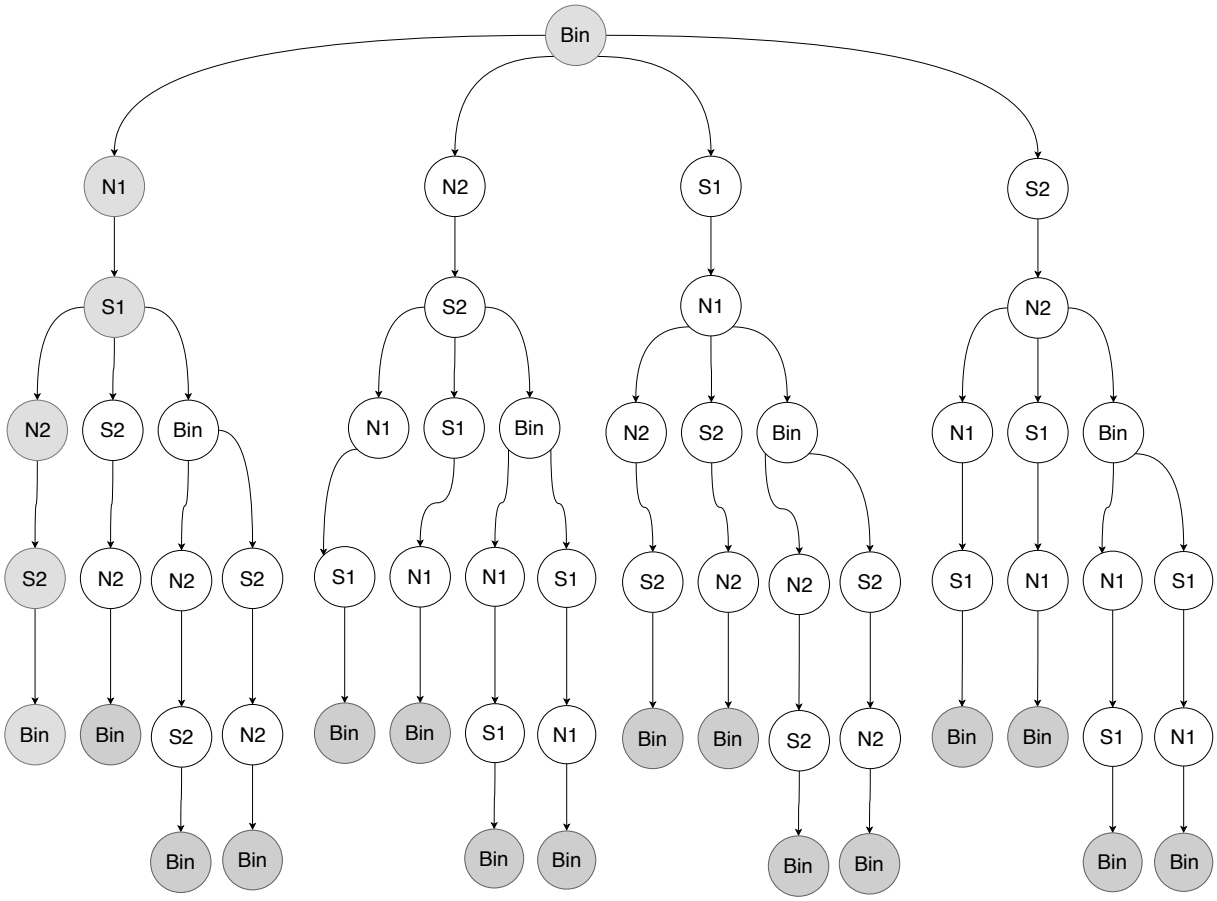


FIGURE 2.10 – Illustration du parcours en profondeur d'un graphe d'état

Les deux illustrations (figures 2.9 et 2.10) expliquent l'influence de la stratégie de parcours sur l'espace d'état parcouru. Évidemment, l'influence de l'un ou l'autre des types de parcours dépend de la propriété vérifiée. Dans le cas de notre exemple, la propriété étant vérifiée sur une feuille du graphe (donc en bout de chemin), le parcours en profondeur d'abord est clairement plus avantageux.

2.5.2 Complexité combinatoire

Le model-checking présente plusieurs limites dont la plus importante est le problème de l'explosion combinatoire. Ce problème se produit au moment de la vérification (phase 3 du processus du MC) lorsque la taille de l'espace d'états, généré au

moment de la vérification des propriétés, excède la quantité de mémoire disponible ce qui rend impossible la vérification. Il faut noter que le nombre d'états augmente de façon exponentielle en fonction de la complexité du système. Même si les modèles sont simples, la taille de l'espace d'état généré pour la vérification peut être importante et on parle du problème de l'explosion combinatoire [32].

La table 2.1, tirée de [92], permet de donner une idée sur la complexité du model-checking (MC) pour certains types de propriétés, en fonction des logiques temporelles utilisées.

Logique	Complexité
CTL	P-complet
LTL	PSPACE-complet
CTL*	PSPACE-complet

TABLE 2.1 – La complexité du model checking pour certaines logiques temporelles [92]

Source typique d'explosion combinatoire : les entrelacements

Une source typique d'explosion combinatoire est les entrelacements. Ces entrelacements sont rencontrés dans la représentation d'un parallélisme d'exécution. La figure 2.11 présente un exemple d'entrelacement. Le système est décrit par deux automates (Automate 1 et Automate 2), et l'état initial est (S1,S3). Lorsque x = 5, deux transitions sont tirables (de S1 à S2 et de S3 à S4) au même moment. Cependant, les automates étant un formalisme avec une sémantique asynchrone, une seule transition est traitée à la fois. Ainsi, le graphe d'états différencie l'ordre de tir de ces deux transitions en générant deux états intermédiaires (S1,S4) et (S2,S3). Puis le système retombe dans un état final commun (S3,S4) quelque soit l'ordre de tir utilisé. La représentation explicite de ces deux chemins est source de complexité potentiellement exponentielle. Or, souvent, ces deux états ne sont pas pertinents pour la propriété à vérifier et on pourrait s'abstenir de représenter explicitement les différents chemins de l'entrelacement.

Nous avons vu, via un exemple, que l'entrelacement peut être une source d'explosion combinatoire. La question qui se pose maintenant : "Est ce qu'il existe des techniques qui permettent de réduire l'explosion combinatoire ?". La section suivante permet de répondre à cette question.

2.5.3 Réduction de l'explosion combinatoire

Plusieurs pistes peuvent être suivies pour tenter de limiter le phénomène d'explosion combinatoire.

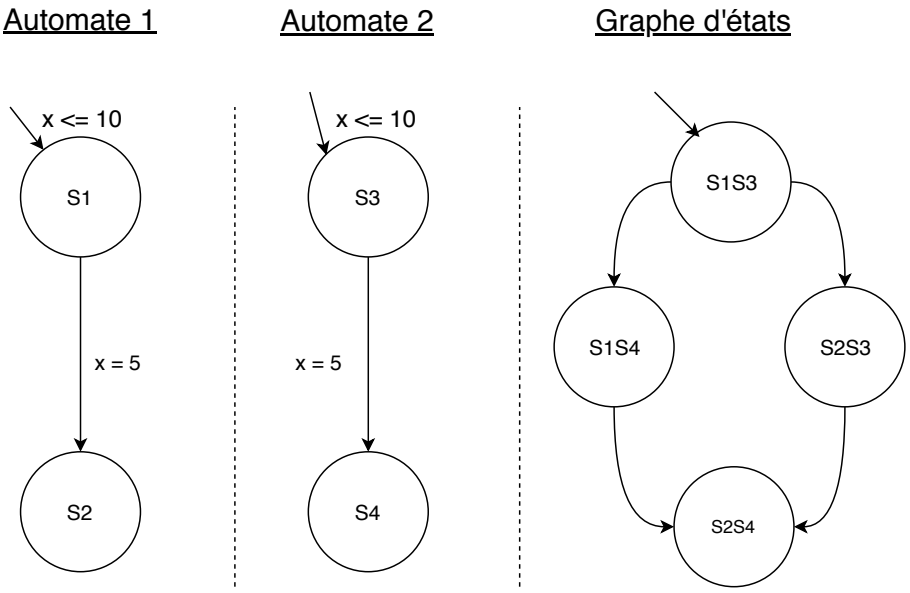


FIGURE 2.11 – Exemple d'entrelacement entre états

2.5.3.1 Par réduction de l'espace d'état

Différentes techniques permettent de limiter l'explosion combinatoire. Nous présentons ici quelques techniques de réductions parmi les plus connues :

- Le model-checking symbolique par les BDD (Binary Decision Diagram) est une technique où les états sont représentés symboliquement par des diagrammes de décision. Cette représentation symbolique permet une représentation plus compacte de l'espace d'états ([39]).
- Les techniques d'ordre partiel [64, 26] sont basées sur l'idée d'éliminer autant que possible les entrelacements inutiles entre les processus lors de la construction de l'espace d'états. Si la différentiation de l'ordre de tir des transitions parallèles est non pertinent pour la propriété à vérifier, on peut s'abstenir de représenter explicitement les différents chemins de l'entrelacement.
- La technique d'exploitation des symétries [71] consiste à exploiter les symétries du système pour construire un espace d'état réduit. Les symétries structurelles du système induisent des symétries sur le graphe de l'espace d'états. La notion de symétrie d'un objet peut se définir comme la stabilité de cet objet vis à vis de certaines transformations.

2.5.3.2 Par optimisation de la modélisation

Il est également important de noter que la modélisation est une étape privilégiée pour la prévention de l'explosion combinatoire. En effet, le choix des abstractions et

des simplifications influe énormément sur la taille de l'espace d'états. Il faut cependant que ces abstractions soit faites tout en garantissant la justesse de la vérification des propriétés (les modèles complets et simplifiés doivent avoir un comportement équivalent au regard de la vérification). Pour cela, et pour être efficace, il est indispensable d'être expert dans la sémantique du langage de modélisation comme dans la technique de vérification sous-jacente.

2.5.3.3 En limitant l'espace d'états exploré

Vérification *on-the-fly* : La technique de vérification à la volée (*on-the-fly*) permet d'analyser le système sans construire tout l'espace d'états ce qui permet de réduire le nombre d'états explorés, et par conséquent réduire le temps de vérification et la mémoire utilisée. Cette réduction n'est cependant effective que pour certaines propriétés, qui ne nécessite pas, par définition, de parcourir l'ensemble du graphe (typiquement l'accessibilité). Cependant, la vérification à la volée est également utile pour les autres propriétés (les invariants par exemple), lorsque la propriété est violée : le premier état qui ne satisfait pas la propriété arrête le processus de vérification, ce qui raccourcit le processus de détection d'erreur lorsqu'il y en a une.

Il y a principalement deux familles d'algorithmes qui permettent d'analyser des systèmes temporisés à la volée. La première, appelée analyse en arrière, consiste à calculer itérativement les prédécesseurs des états que l'on cherche à atteindre et à tester si un état initial se trouve dans l'ensemble calculé. La seconde, appelée analyse en avant, consiste à calculer itérativement les successeurs des états initiaux et à tester si l'état que l'on cherche à atteindre a déjà été calculé.

Vérification en largeur ou profondeur d'abord : Même dans le cas où la méthode de vérification ne nécessite de parcourir qu'une partie de l'espace d'état, le choix de la stratégie de parcours (et de construction) du graphe d'états du système décrit à la sous-sous-section 2.5.1.2 peut également avoir une influence sur les performances globales de la vérification.

Nous allons maintenant voir quelques outils de MC.

2.6 Outils de model-checking

Dans cette section, nous donnons un aperçu sur quelques outils de model-checking. Ensuite, nous présentons l'outil UppAal choisi pour le travail de recherche présenté dans ce mémoire.

2.6.1 Aperçu des outils de model-checking

Il existe de nombreux outils de model-checking. Une liste étendue des logiciels de vérification est disponible via l'url présentée en pieds de page [2]. Le lecteur intéressé par des descriptions plus détaillées des outils peut se reporter aux articles [21, 60, 94]. D'un point de vue pratique, nous pouvons classer les outils de vérification en 4 catégories d'usage : ceux dédiés à l'analyse de code (BLAST, CPAchecker), ceux voués au MC ordinaire (SPIN, TAPAs, TLA+, NuSMV), ceux permettant le MC probabiliste (CADP, MRMC, PAT, PRISM), et enfin les outils pour le MC temporisé (DREAM [3], LTSmin [4], mCRL2 [5], TAPAAL [6], UppAal [7], ROMEO [8], HYTEC [9], TINA [10], Kronos [11]). Nous nous intéressons à ces derniers puisque certains des problèmes d'agriculture de précision auxquels nous nous intéressons sont des problèmes temporisés.

TAPAAL, TINA et ROMEO sont utilisés pour la validation et la vérification des systèmes temps réel modélisés en réseaux de Petri temporel (RPT). Hytech permet d'analyser des systèmes décrits comme des Automates Hybrides Linéaires (AHL) [31]. LTSmin et mCRL2 sont utilisés pour la vérification des systèmes modélisés en algèbre de processus. Les algèbres de processus sont une famille de langages formels permettant de modéliser les systèmes (informatiques) concurrents ou distribués. Kronos a été beaucoup utilisé par la communauté scientifique avant l'année 2009 mais depuis sa maintenance n'est plus assurée par son équipe de développement. Un avantage spécifique de Kronos est qu'il peut fournir (option de vérification) toutes les traces qui vérifient une propriété donnée. L'outil DREAM a été très peu utilisé et il n'a pas une interface graphique utilisateur conviviale.

UppAal est, aujourd'hui, l'outil d'analyse de systèmes temporisés le plus abouti : il est régulièrement mis à jour et une interface graphique conviviale assure une prise en main aisée du logiciel. Il est également énormément utilisé par la communauté scientifiques [57, 67, 110, 93]. Les modèles manipulés par l'outil sont exprimés dans une variante équivalente du formalisme des automates temporisés, qui sont étendus avec des variables entières [12].

Nous avons estimé que UppAal était bien adapté au travail de recherche présenté dans ce mémoire, et en particulier qu'il permettrait de démontrer l'intérêt de faire appel aux techniques de model-checking pour vérifier des systèmes avec une com-

2. https ://cps-vo.org/group/verification_tools/wiki
3. http ://dre.sourceforge.net/
4. http ://ltsmin.utwente.nl/
5. https ://www.mcrl2.org/web/user_manual/index.html
6. http ://www.tapaal.net/
7. http ://www.UppAal.org/
8. http ://romeo.rts-software.org/
9. https ://ptolemy.berkeley.edu/projects/embedded/research/hytech/
10. http ://www.laas.fr/tina/
11. http ://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos/

posante spatiale, dans le cadre de l'agriculture de précision. En effet, l'utilisation de variables entières facilite en pratique la modélisation. La représentation en réseaux d'automates temporisés synchronisés par évènements permet une modularité de modélisation qui est adaptée à la représentation conjuguée des systèmes biologiques et techniques impliqués par l'agriculture de précision. Par exemple, le modèle développé au chapitre 5 représente d'une part l'état des rangs de végétation, avec un automate par rang, et d'autre part le comportement de la machine de récolte.

Par ailleurs, l'outil UppAal, mature au plan technique, est encore largement utilisé par la communauté scientifique, avec un soutien aux utilisateurs. C'est un atout pratique important pour cette recherche.

TABLE 2.2 – Classification des outils de vérification

Outil de MC	Classe du MC	Lang. de modélisation	Lang. de propriété	GUI
BLAST	analyse du code	C	Monitor automata	Non
CPAchecker	analyse du code	C	Monitor automata	Oui
CADP	probabiliste	-	-	Oui
MRMC	probabiliste	-	-	Non
PAT	probabiliste	-	-	Oui
PRISM	probabiliste	-	-	Oui
SPIN	ordinaire	-	-	Oui
TAPAs	ordinaire	-	-	Oui
TLA+	ordinaire	-	-	Oui
NuSMV	ordinaire	-	-	Non
DREAM	temps réel	AT	Monitor automata	Non
LTSmin	temps réel	Algèbre de processus	$\mu - calculus$, LTL, CTL	Non
HYTEC	temps réel	AHL	-	-
Kronos	temps réel	AT	TCTL	-
TAPAAL	temps réel	RPT	TCTL	Oui
TINA	temps réel	RPT	-	Oui
ROMEO	temps réel	RPT	TCTL	Oui
mCRL2	temps réel	Algèbre de processus	$\mu - calculus$	Oui
UppAal	temps réel	AT	TCTL	Oui

De plus, une comparaison d'outils de vérification a été faite pour une application sur les systèmes de contrôle des trains dans [94]. Cette étude compare les performances de dix environnements de vérification formels différents (UMC, SPIN, NuSMV/nuXmv, mCRL2, CPN Tools, FDR4, CADP, TLA+, UppAal et ProB) en répondant à une propriété de sûreté. Cette étude démontre l'efficacité et la supériorité d'UppAal ce qui nous a conforté dans notre choix. Il faut assumer que LTSmin et

DREAM permettent de vérifier des modèles conçus avec l'outil UppAal. Idéalement, il aurait été intéressant de comparer les performances entre les trois outils UppAal, LTSmin et DREAM, pour les cas étudiés dans ce mémoire, ce que faute de temps, nous n'avons pu faire, nous avons cependant choisi UppAal sans cette comparaison. D'autant plus que les automates temporisés et UppAal ont déjà été utilisés dans les domaines de l'agriculture et de la gestion des écosystèmes (par exemple, dans [70], [86], [57]), ce qui nous conforte encore dans ce choix.

Maintenant, nous allons présenter en détails l'outil UppAal.

2.6.2 UppAal

L'outil UppAal a été développé en 1995, conjointement par l'université d'Uppsala (Upp) en Suède et l'université d'Aalborg (Aal) au Danemark, d'où son nom [88]. C'est un environnement intégré d'outils pour la modélisation, la validation et la vérification de systèmes en temps réel modélisés comme des réseaux d'automates temporisés. Les automates temporisés d'UppAal sont une variante des automates temporisés d'Alur et Dill, étendus avec des variables entières, des types de données structurés, des fonctions définies par l'utilisateur et la synchronisation des canaux. Notons qu'il y a deux types de canaux dans UppAal :

- Les canaux binaires : un émetteur envoie le signal m ! et un unique récepteur se synchronise avec lui par le signal complémentaire m ?. Les transitions de ces deux automates étiquetées par m ! et m ? sont tirées en même temps. Si, à partir d'une configuration, plusieurs couples de transitions peuvent être synchronisées, un choix est fait de façon non déterministe. La synchronisation par canaux binaires peut conduire à des blocages si un automate envoie un signal m ! mais qu'aucun autre automate ne peut le recevoir (par exemple si la garde de la transition étiquetée par m ? est fausse).

- Les canaux multiples (broadcast) : les transitions étiquetées par m ? et m ! sont synchronisées, mais cette fois avec un émetteur et un nombre quelconque de récepteurs, qui peut être égal à zéro. Ainsi, une synchronisation par canaux multiples ne conduit jamais à un blocage du modèle.

Les états des automates temporisés d'UppAal peuvent être étiquetés de 2 façons :

- L'étiquette U pour *Urgent* : un tel état est équivalent au fait de remettre à zéro une horloge x avant d'arriver dans cet état et d'ajouter la garde x ≤ 0 sur les transitions d'action sortant de cet état. L'automate doit donc le quitter en un temps nul. Les délais dans les états urgents sont toujours égaux à zéro. Dans la figure 2.12 l'automate P est équivalent à l'automate Q. Cette portion de modèle correspond par exemple au fonctionnement d'un système de communication qui reçoit des paquets sur le canal a et les envoie immédiatement sur le canal b.

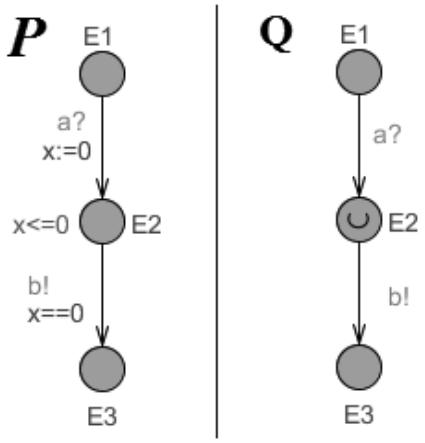


FIGURE 2.12 – Exemple pour l'étiquette urgent

- L'étiquette C pour *Committed* : cette notion est plus restrictive que la notion d'état urgent. L'automate doit quitter un état marqué par C immédiatement après son arrivée en empruntant une des transitions sortantes (s'il y en a plusieurs). Cette étiquette permet de diminuer les cas d'entrelacement à examiner par le model checkeur. En effet, un réseau d'automates peut présenter plusieurs transitions tirables à un même moment, même pendant un intervalle de temps de durée nulle. Lorsque l'état courant dans un des automates du réseau est *Committed*, une de ses transitions sera tirée avant toute autre transition dans un autre automate, y compris celle sortant d'un état marqué *Urgent*. Cela permet de représenter un niveau de priorité dans le tir des transitions.

Les propriétés supportées par UppAal sont principalement des propriétés d'atteignabilité, de vivacité ou d'états bloquants [12]. Le connecteur temporel G est représenté par [] et le connecteur temporel F est représenté par <>. La logique utilisée par UppAal est uniquement un fragment de TCTL qui ne permet pas de vérifier l'ensemble des formules TCTL. Les connecteurs ne pouvant pas être tous imbriqués, seules les propriétés suivantes sont vérifiées par UppAal : A[]p, A<>p, E<>p, E[]p, p –> q et A[] not deadlock [12]. Il faut noter que si UppAal accepte TCTL alors il est possible de vérifier des formules CTL.

UppAal implémente la technique de vérification à la volée (*on-the-fly*).

Il propose un ensemble d'options pour contrôler le comportement du MC au moment de l'exploration de l'espace d'état (largeur ou profondeur d'abord, génération de traces,..). Ces options influent sur l'ordre dans lequel l'espace d'états est exploré, le nombre d'états explorés, la quantité de mémoire allouée au processus de vérification et le temps de vérification. A la différence de l'outil Kronos, UppAal ne fournit

qu'une seule trace, il ne possède pas une option qui permet de donner toutes les traces. D'autre part, il ne fournit pas malheureusement le graphe d'états généré au moment de la vérification.

Le site internet d'UppAal [12] et notamment l'onglet *Web Help* permet d'accéder à un manuel d'utilisation qui contient toutes les informations nécessaires avec des explication détaillées, ce qui facilite la découverte et la maîtrise de l'outil.

Dans plusieurs des cas étudiés dans ce mémoire, nous souhaitions pouvoir accéder à la trace d'un chemin d'exécution optimal selon un critère de coût. Avec UppAal, il serait possible de modéliser explicitement le coût par une variable entière et ensuite de chercher par dichotomie la valeur optimale du coût. Ce serait cependant une méthode peu efficace. La recherche par dichotomie oblige à relancer plusieurs fois la vérification. D'autre part, il n'existe aucune méthode dans UppAal qui permette de spécifier une variable comme monotone. La définition d'un coût comme une variable ordinaire conduit donc a priori à augmenter l'espace d'états. Il existe heureusement une variante d'UppAal qui répond à notre attente et que nous détaillons dans la section suivante.

2.7 Problématique d'analyse d'accessibilité à coût optimal (CORA)

2.7.1 Définition générale du problématique d'analyse d'accessibilité à coût optimal (CORA)

CORA est l'abréviation de Cost Optimal Reachability Analysis (en français : analyse d'accessibilité à coût optimal). Elle adresse le problème de trouver le coût minimum pour atteindre un état cible [87, 17]. Étant donné un réseau d'Automates Temporisés de coûts, CORA représente le problème de trouver le coût minimum pour atteindre un état cible dans l'espace d'états de ce modèle, si cet état est accessible [25]).

La notion de coût optimal est souvent utilisée dans les problèmes d'ordonnancement et de planification. Ceux-ci relèvent des domaines de l'Intelligence Artificielle et de la Recherche Opérationnelle. Parmi les méthodes de résolution de ce type de problème, on peut typiquement trouver la programmation linéaire simple ou par des techniques de propagation de contraintes. Une approche alternative consiste à utiliser la vérification formelle et à rechercher, parmi l'ensemble des chemins atteignables dans le graphe d'états généré par la vérification du système, un chemin de coût optimal. L'objectif des concepteurs de CORA a été d'établir des liens avec d'autres disciplines et communautés qui s'intéressent à ces problèmes car la planification et la synthèse de contrôleur sont des domaines assez proches ([105], voir chapitre 3).

12. *http* : *//www.UppAal.org*

Certaines des problématiques qu'on veut adresser sont des problématiques CORA, par exemple le problème de récolte sélective qui sera abordé dans le chapitre 5 et le problème de pulvérisation de précision qui sera abordé dans le chapitre 4. Nous verrons que ces problèmes peuvent être appréhendés comme des problèmes de planification et d'ordonnancement.

La question qui se pose maintenant est : comment l'outil UppAal peut-il répondre à une problématique de vérification d'une propriété d'atteignabilité donnée avec optimisation ?

L'algorithme standard de vérification de UppAal ne comprend pas la possibilité d'effectuer une recherche d'optimalité. Dans les options de génération d'une trace, on peut obtenir la trace la plus courte en nombre de transitions, ou la trace la plus rapide en durée, qui sont des traces pertinentes à analyser, mais cela ne permet pas d'optimiser sur un critère choisi.

Il est donc nécessaire de se pencher sur une extension spécifique de la suite logicielle UppAal : UppAal-CORA, qui est spécifique pour la résolution de ce genre de problème. UppAal-CORA utilise un formalise de modélisation spécifique, les automates temporisés de coût, que nous introduisons dans la section suivante. Puis, nous présenterons l'outil UppAal-CORA et ses principes algorithmiques. Enfin, nous introduirons l'outil memtime utilisé pour l'étude des performances des processus de vérification/optimisation.

2.7.2 Formalisme des Automates Temporisés de Coût (ATC)

En 2001 et d'après [59], Behrmann et al. [14] et, de manière indépendante, Alur et al. [5] ont introduit les Automates Temporisés de Coût (ATC). Behrmann et al. appellent les ATC *Priced Timed Automata* (PTA) et Alur et al. appellent les ATC *Weighted Timed Automata* (WTA). Les deux dénominations désignent en fait le même formalisme. Les ATC peuvent être vus comme des automates temporisés (AT) avec une variable réelle non-négative spécifique appelée coût. Le coût désigne une variable monotone croissante sur les réels non négatifs (elle ne diminue jamais lorsque le temps s'écoule).

Donnons une définition formelle d'un automate temporisé de coût [104] : soit C un ensemble d'horloges, $B(C)$ l'ensemble des formules qui sont des conjonctions de contraintes atomiques sur ces horloges de la forme $x \bowtie n$ et $x - y \bowtie m$ pour $x, y \in C$, $\bowtie \in \{<, \leq, =, \geq, >\}$, m et n des entiers et A un ensemble d'actions. Un automate temporisé de coût sur C et A est un 5-uplet (L, E, l_0, inv, $cout$) où L est un ensemble d'états, $l_0 \in L$ est l'état initial, $E \subseteq L \times B(C) \times Ax2^C \times L$ est un ensemble de transitions entre états avec un état source, une garde, une action, l'ensemble d'horloge à initialiser et un état destination. $inv : L \to B(C)$ assigne des invariants aux états et $cout : L \cup E \to N$ assigne des informations sur l'évolution du coût aux transitions et aux états.

Dans un automate temporisé de coût, on exprime l'évolution du coût d'une part sur les états et d'autre part sur les transitions :

1. La première façon est d'exprimer l'évolution du coût comme un taux d'évolution dans l'invariant d'un état de l'automate. Comme le montre la figure 2.13, le taux d'évolution du coût (noté *coût′* dans UppAal) dans l'état *Initial* est égal à 2. Comme l'automate reste 3 unités de temps dans cet état, le coût augmente de 6. Dans le langage de description d'automate temporisé de coût de l'outil UppAal-CORA, le taux d'évolution du coût dans l'invariant de l'état est exprimé par l'expression $cost' == c$ avec $c \in \mathbb{R}^+$.

2. La deuxième façon est d'exprimer l'évolution du coût comme un entier associé à une transition discrète de l'automate. Cet entier indique le coût de tir de cette transition, et donc un incrément apporté au coût total. Par exemple, dans la figure 2.13, le coût s'incrémente de 5 lors du tir de la transition. Le coût total pour atteindre l'état *Final* dans l'automate exemple est donc égal à $11(6+5)$. Dans UppAal-CORA, le coût est exprimé par l'expression $cost+ = c$ avec $c \in \mathbb{R}^+$. Il s'exprime donc au travers de sa mise à jour dans la transition de l'automate. Il faut noter que la variable de coût ne peut qu'augmenter.

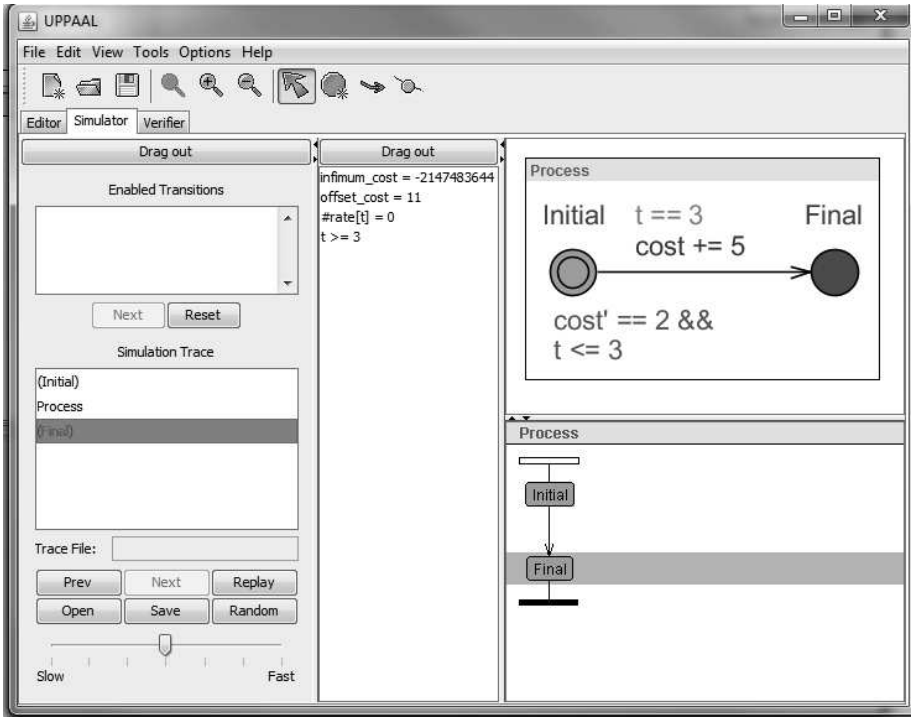


FIGURE 2.13 – Exemple d'utilisation du coût

On peut noter que si le système est représenté par un réseau d'automates, le coût total est l'addition des coûts de tous les automates. Un exemple de tel système est représenté dans le chapitre 4 dans la section 4.3.3.

Le formalisme des ATC est très expressif. Il permet à l'algorithme de model-checking de rechercher efficacement le chemin optimal en plus de la vérification d'atteignabilité. L'outil UppAal-CORA permet donc de modéliser avec les ATC et de combiner vérification et optimisation.

2.7.3 UppAal-CORA

2.7.3.1 Présentation de l'outil UppAal-CORA

UppAal-CORA ([16],[17]) est une variante de UppAal pour l'analyse d'accessibilité à coût optimal pour les automates temporisés de coût. Il a été développé en 2004 par l'équipe UppAal dans le cadre des projets VHS (Verification of Hybrid Systems) et AMETIST (Advanced MEthod for TImed SysTems). UppAal-CORA prend en entrée un ensemble d'automates temporisés de coût et une requête exprimée en une restriction de la logique TCTL. Il permet de vérifier une propriété donnée, et fournit une trace d'exécution qui optimise le coût pour atteindre cette propriété. UppAal-CORA a été utilisé avec succès pour des problèmes de planification et de routage dans plusieurs études de cas ([16]) : planification de tâches, problème d'atterrissage d'aéronef, problème de tournées de véhicules avec fenêtres de temps, etc.

Dans la suite, les principes algorithmiques de UppAal-CORA sont présentés.

2.7.3.2 Ses principes algorithmiques

L'algorithme de vérification de principe "CORA" est connu sous le nom de *abstract algorithm for the minimal-cost reachability problem* (algorithme abstrait pour le problème d'accessibilité de coût minimal) [15]. Il explore l'ensemble de l'espace d'états d'un modèle exprimé en automates temporisés de coût donné pour trouver un chemin à coût optimal vers un état vérifiant la propriété spécifiée dans la requête d'atteignabilité. L'algorithme abstrait pour le problème d'accessibilité de coût minimal est défini comme le montre l'algorithme 1.

Il faut noter que l'algorithme 1 se base sur le concept algorithmique de *branch and bound*. Avant d'expliquer cet algorithme, nous avons besoin de donner une définition d'un état symbolique de coût. Soit (l, C) est un état symbolique de coût où l est un état dans un automate temporisé de coût et C est une fonction de coût qui associe à chaque évaluation d'horloge une valeur de coût.

L'algorithme 1 utilise deux structures de données WAITING et PASSED pour stocker d'une part les états qui doivent être explorés et d'autre part les états qui ont été déjà explorés. Initialement, PASSED est vide et WAITING contient l'état initial (l_0, C_0). A chaque itération (à l'intérieur de la boucle `while`), l'algorithme

procède en sélectionnant un état (l, C) de WAITING [13] et en testant, d'une part si cet état vérifie la propriété ciblée, et d'autre part si la meilleure valeur de coût (selon la valuation d'horloge) de cet état n'est pas plus petite que la borne supérieure du coût optimal connue COST. Dans ce cas, cette borne supérieure est mise à jour. L'algorithme ajoute ensuite, s'il n'a pas déjà été exploré avec un coût inférieur, (l, C) à l'ensemble PASSED et ses successeurs à l'ensemble WAITING. L'algorithme se termine lorsque WAITING est vide, c'est-à-dire lorsqu'il ne reste plus d'états à examiner. Cet algorithme de principe explore l'intégralité des états de l'automate analysé.

Algorithm 1 algorithme abstrait pour le problème d'accessibilité de coût minimal [15]

```
1: function CORA Algorithm()
2:     COST := ∞
3:     PASSED := ∅
4:     WAITING := {(l_0, C_0)}
5:     while WAITING ≠ ∅ do
6:         select (l, C) from WAITING // basé sur une stratégie de branchement
7:         if (l, C) ⊨ φ and min(C) < COST then
8:             COST := min(C)
9:         end if
10:        if
11:            for all (l, C') in PASSED : C' ⋢ C do then
12:                add(l, C) to PASSED
13:                for all (m, D) such that (l, C) ⇝ (m, D) do
14:                    add (m, D) to WAITING
15:                end for
16:            end for
17:        end if
18:    end while
19:    return COST
20: end function
```

L'algorithme de vérification de UppAal-CORA se base sur l'algorithme de principe CORA expliqué ci-dessus. L'inconvénient majeur de cet algorithme de principe est qu'il nécessite une recherche de l'espace d'état complet avant de pouvoir déterminer le coût minimum permettant d'atteindre un état vérifiant une propriété donnée. En effet, en sélectionnant un état de WAITING, la sélection se base sur une stratégie de branchement, c'est-à-dire d'ordre d'exploration de l'arbre des états. Diverses stratégies de parcours sont incluses dans UppAal-CORA telles que la largeur d'abord, la profondeur d'abord, le meilleur d'abord (best first), etc. Si la stratégie de branche-

13. La sélection se base sur une stratégie de branchement

ment est, par exemple, la largeur d'abord, alors rien ne garantit que le premier état vérifiant la propriété donnée appartient à une solution optimale. Potentiellement, tout l'espace d'états peut être exploré pour déterminer le coût minimum. Dans le cas où la stratégie de branchement est le meilleur d'abord, l'état avec la plus petite valeur de la variable de coût est recherché en premier. Si l'état obtenu par exploration "best-first" réalise l'objectif d'atteignabilité, alors le chemin trouvé est optimal. De façon générale, la méthode "best first" permet de limiter considérablement le nombre d'états explorés.

L'algorithme de vérification de UppAal-CORA utilise certaines techniques pour réduire la recherche de l'espace d'états. En effet, à partir d'un état donné pour lequel on a évalué un coût partiel, on a souvent une idée du coût minimum restant pour atteindre un objectif. La somme du coût partiel et de ce coût minimum restant fournit une borne inférieure du coût total du chemin d'exécution en cours d'évaluation. Cette borne inférieure peut être comparée au minimum de tous les coûts totaux obtenus jusqu'ici lors du parcours du graphe d'états.

Cette information du coût restant est utilisée à la fois pour réduire le nombre d'états à évaluer et pour rechercher d'abord les états les plus prometteurs [16]. Dans UppAal-CORA, l'information sur la borne inférieure du coût restant est stockée dans une méta variable appelée *remaining*. Les méta-variables sont stockées dans le vecteur d'état, mais ne sont pas considérées comme faisant partie de l'état. Ainsi, deux états qui ne diffèrent que par des méta-variables sont considérés égaux [14]. Dans l'outil UppAal-CORA, la variable *remaining* est déclarée en utilisant l'instruction *meta int remaining*; dans le système d'automates. Les instructions d'affectation de la variable *remaining* sont associées aux transitions dans l'automate. Il faut noter que pour utiliser *remaining* de façon efficace, il faut déterminer une bonne estimation du coût, tout en garantissant qu'il s'agit bien d'une borne inférieure. Dans le cas où cette propriété de borne inférieure ne serait pas garantie, il y aurait alors un risque de ne pas évaluer un chemin pourtant optimal et donc de fournir un coût minimal qui n'est pas l'optimal réel.

UppAal-CORA dispose de plusieurs techniques de choix des successeurs à évaluer dans sa recherche en avant. Celle qui nous intéresse le plus est la stratégie de recherche "Best First" que nous utilisons ici en exploitant la fonction de *remaining*. Expliquons le principe avec un exemple :

Soit (l, C) un état de coût symbolique et $REM(l, C)$ une limite inférieure sur le coût restant de tous les états ayant un état l. En combinant le coût minimum $min(C)$ d'un état de coût symbolique (l, C) avec l'estimation du coût restant $REM(l, C)$, on peut baser l'ordre d'exploration sur la somme $min(C) + REM(l, C)$. L'utilisation d'informations sur le coût restant peut également réduire le nombre d'états recherchés avant qu'une solution optimale ne soit atteinte (voir un exemple dans le chapitre 5 la sous-section 5.5.3).

14. `http://cl-informatik.uibk.ac.at/teaching/ss07/vmc/material/uppaal-4-language.pdf`

Comme l'algorithme de *branch and bound* permet d'identifier et d'élaguer des parties de l'espace d'états, cette technique est évidemment très utile pour résoudre un problème de type CORA pour un système complexe. Cependant, la fonction pour estimer le coût restant nécessite une grande expertise, une forte intuition et une compréhension approfondie du système. Cette fonction est d'autant plus importante qu'une estimation fausse du coût restant peut supprimer la solution optimale, et une estimation grossière ne permet pas de beaucoup filtrer l'espace d'états. C'est de la responsabilité de l'utilisateur de garantir que l'estimation est bien une limite inférieure afin de s'assurer que la solution optimale n'est pas supprimée, et qu'elle est assez fine pour bien réduire l'exploration.

Dans la suite, un exemple d'exploration de l'espace d'état avec et sans remaining est donné.

2.7.3.3 Exemple d'exploration de l'espace d'état avec et sans remaining

Pour illustrer le fonctionnement de l'algorithme de vérification de UppAal-CORA, nous reprenons l'exemple de la version simplifiée du problème de récolte décrit dans la section 2.5. Le graphe d'états de l'exemple illustré dans la figure 2.8 représente tous les chemins possibles du robot dans la parcelle. A chaque noeud du graphe, on doit lui associer une valeur du coût. Le coût dans notre exemple représente la distance parcourue par le robot. Comme il y a plusieurs noeuds qui ont le même nom (par exemple le noeud "S1" se répète 10 fois) et pour éviter les confusions, tous les noeuds du graphe d'état de la figure 2.8 sont renommés dans la figure 2.14 par des noms uniques allant de $E0$ à $E60$.

Dans la suite, nous présentons deux stratégies d'exploration de graphe : la première n'utilise pas la fonctionnalité de remaining et la deuxième l'utilise.

A) Stratégie d'exploration de l'espace d'état sans remaining

Au début, la structure de données PASSED est vide et la structure de données WAITING contient $E0$, avec un coût nul. Chaque élément de WAITING ou de PASSED est représenté par un couple (X, Y) avec X le nom de l'élément et Y son coût. L'algorithme de vérification de UppAal-CORA commence la première itération : il sélectionne $E0$ de la liste WAITING et comme $E0$ n'est pas un état final alors il passe dans la liste PASSED et ses successeurs $E1$, $E2$, $E3$ et $E4$ sont ajoutés à la liste WAITING. $E0$ est supprimé de la liste WAITING. Le coût de chaque élément de WAITING est calculé (on rappelle que la figure 2.7 permet de représenter la parcelle ainsi que les coûts des chemins). Après la première itération, on a PASSED $= \{(E0, 0)\}$ et WAITING $= \{(E1, 210), (E2, 240), (E3, 20), (E4, 30)\}$.

L'algorithme passe à la deuxième itération et il sélectionne $E3$ de WAITING parce qu'il a le coût le plus petit. $E3$ n'est pas un état final alors il passe dans la liste PASSED et son successeur $E7$ est ajouté à la liste WAITING. Le coût de $E7$

est calculé : c'est le coût de $E3$ (20) + le coût de la transition entre $E3$ et $E7$ (180). $E3$ est supprimé de la liste WAITING. Ainsi de suite, jusqu'à atteindre le noeud $E45$ qui est un état final de coût égal à 430. Le tableau 2.3 présente le coût de tous les noeuds du graphe d'état. La couleur rouge correspond aux états finaux. La couleur jaune correspond aux états explorés et la couleur orange correspond à l'état qui vérifie notre objectif. On peut bien voir qu'aucun chemin menant à un noeud final n'a un coût inférieur à celui trouvé par notre algorithme (seul le noeud E49 a un coût équivalent).

TABLE 2.3 – Le coût de tous les noeuds du graphe d'état

Noeud	Coût	Noeud	Coût	Noeud	Coût
E0	0	E21	780	E41	820
E1	210	E22	590	E42	830
E2	240	E23	650	E43	840
E3	20	E24	440	E44	650
E4	30	E25	800	E45	430
E5	390	E26	620	E46	820
E6	420	E27	660	E47	830
E7	200	E28	470	E48	620
E8	210	E29	400	E49	430
E9	600	E30	580	E50	810
E10	410	E31	650	E51	840
E11	410	E32	440	E52	650
E12	620	E33	410	E53	860
E13	440	E34	600	E54	860
E14	450	E35	660	E55	860
E15	220	E36	470	E56	860
E16	400	E37	810	E57	860
E17	410	E38	830	E58	860
E18	230	E39	830	E59	860
E19	420	E40	620	E60	860
E20	450				

Quand on atteint $E45$, on a :
PASSED = $\{(E0, 0), (E3, 20), (E4, 30), (E7, 200), (E8, 210), (E1, 210), (E15, 220), (E18, 230), (E2, 240), (E5, 390), (E16, 400), (E29, 400), (E17, 410), (E33, 410), (E10, 410), (E11, 410), (E19, 420), (E6, 420), (E45, 430)\}$.
et WAITING = $\{E20, 450), (E9, 600), (E30, 580), (E31, 650), (E32, 440), (E49, 430), (E22, 590), (E23, 650), (E24, 440), (E34, 600), (E12, 620), (E13, 440), (E14, 450)\}$.
Les deux listes sont ordonnées par l'ordre d'exploration des noeuds. Les noeuds de la liste WAITING ont un coût supérieur au coût optimal de $E45$ donc ils ne seront

pas explorés. Dans la figure 2.14, les noeuds en couleur jaune sont les noeuds explorés (noeuds de la liste PASSED) et les noeuds barrés sont les noeuds de la liste WAITING. Les branches des noeuds barrés ne seront pas explorées d'où la réduction de l'espace d'état généré.

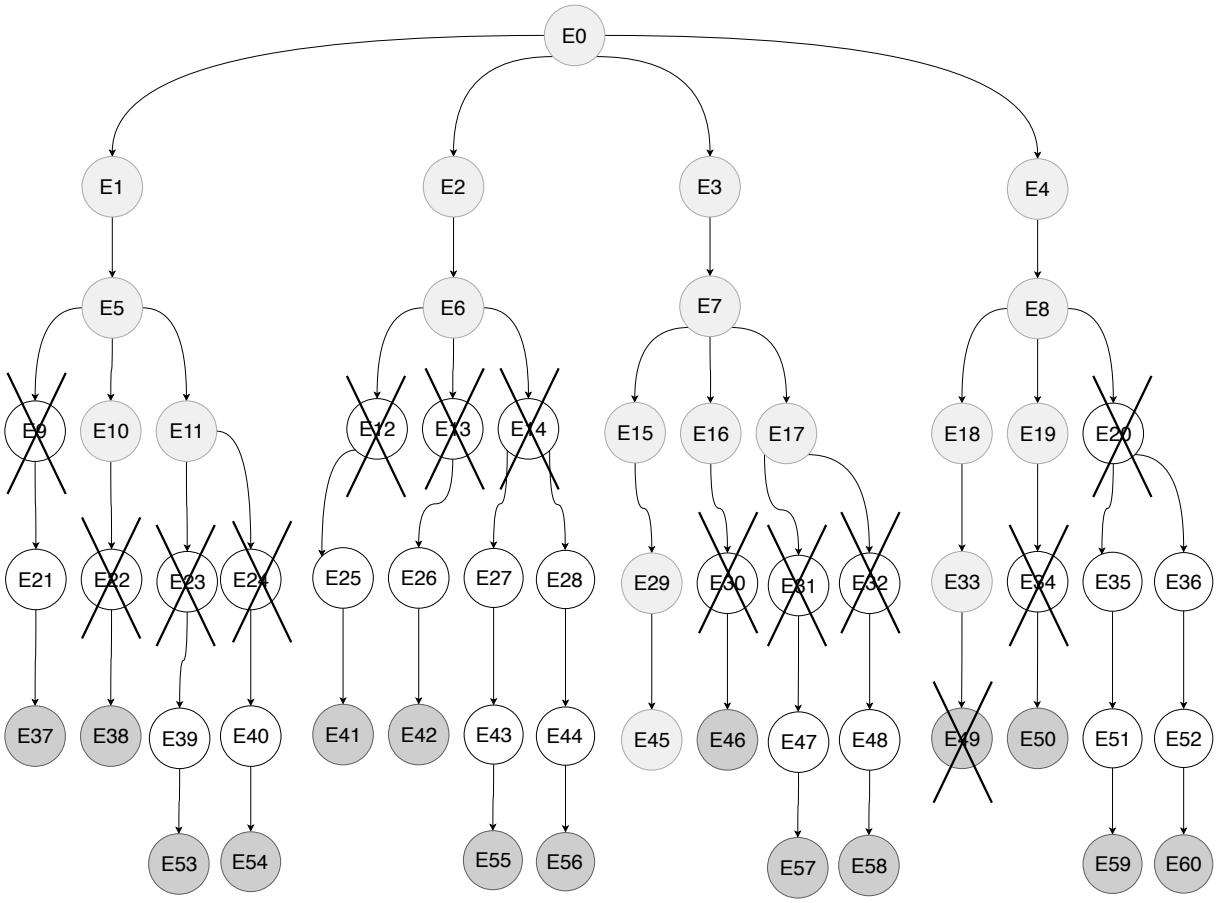


FIGURE 2.14 – Illustration du parcours de l'algorithme "CORA" des noeuds sans remaining

Dans la suite, la deuxième stratégie d'exploration en utilisant la fonction remaining sera détaillée.

B) Stratégie d'exploration de l'espace d'état avec remaining

La fonction remaining calcule le coût restant "Remaining" pour chaque noeud du graphe, elle est définit comme suit :

$$Remaining = TR + TABR + TRRB + TCR. \tag{2.1}$$

avec : TR le Temps de Récolte des rangs non encore récoltés, TABR le Temps minimum d'Aller de l'emplacement *bin* à un Rang non encore récolté, TRRB le Temps minimum de Retour d'un Rang non récolté à l'emplacement *bin* et TCR le Temps de Circulation entre les Rangs. Ces données sont calculées en fonction du graphe donné figure 2.7.

Au début, la structure de données PASSED est vide et la structure de données WAITING contient $E0$ dont la valeur du coût 0 et la valeur du coût restant "*Remaining*" est 460. Chaque élément de WAITING ou de PASSED est représenté par un triple (X,cout,remain) avec X le nom de l'élément, *cout* son coût et *remain* son "*Remaining*". L'algorithme de vérification de UppAal-CORA commence la première itération et il sélectionne $E0$ de la liste WAITING et comme $E0$ n'est pas un état final alors il passe à la liste PASSED et ses successeurs $E1$, $E2$, $E3$ et $E4$ sont ajoutés à la liste WAITING. $E0$ est supprimé de la liste WAITING. Le coût et le "*Remaining*" de chaque élément de WAITING sont calculés. Après la première itération, on a PASSED = $\{(E0, 0, 420)\}$ et WAITING = $\{(E1, 210, 410), (E2, 240, 400), (E3, 20, 410), (E4, 30, 400)\}$.

L'algorithme passe à la deuxième itération et il sélectionne $E3$ de WAITING parce qu'il a la valeur de la somme du coût et de "*Remaining*" la plus petite. $E3$ n'est pas un état final alors il passe à la liste PASSED et son successeur $E7$ est ajouté à la liste WAITING. Le coût et le "*Remaining*" de $E7$ sont calculés. $E3$ est supprimé de la liste WAITING. Ainsi de suite, jusqu'à atteindre le noeud $E45$ qui est un état final. On a la liste PASSED = $\{(E0, 0, 420), (E3, 20, 410), (E4, 30, 400), (E7, 200, 230), (E8, 210, 220), (E15, 220, 210), (E18, 230, 200), (E29, 400, 30), (E33, 410, 20), (E45, 430, 0)\}$ et la liste WAITING = $\{(E1, 210, 410), (E2, 240, 400), (E16, 400, 210), (E17, 410, 240), (E19, 420, 200), (E20, 450, 220), (E49, 430, 0)\}$. Les deux listes sont ordonnées par l'ordre d'exploration des noeuds. Les noeuds de la liste WAITING ont un coût total (cout + "*Remaining*") supérieur au coût optimal de $E45$ donc ils ne seront pas explorés. Dans la figure 2.15, les noeuds en couleur jaune sont les noeuds explorés (noeuds de la liste PASSED) et les noeuds barrés sont les noeuds de la liste WAITING. Les branches des noeuds barrés ne seront pas explorées d'où la réduction de l'espace d'état généré.

En conclusion, pour l'exemple étudié, les illustrations du parcours de l'algorithme "CORA" des noeuds avec et sans remaining expliquent comment l'algorithme "CORA" recherche d'abord les états les plus prometteurs et prouvent que cette recherche permet de réduire le nombre d'états recherchés pour atteindre la solution optimale. Il faut toujours noter que la définition de la fonction de remaining est très importante car une estimation fausse peut élaguer la solution optimale et une estimation peu serrée ne réduit pas le suffisamment l'espace d'état exploré.

Dans la section suivante, l'outil utilisé pour évaluer les performances de résolution est décrit.

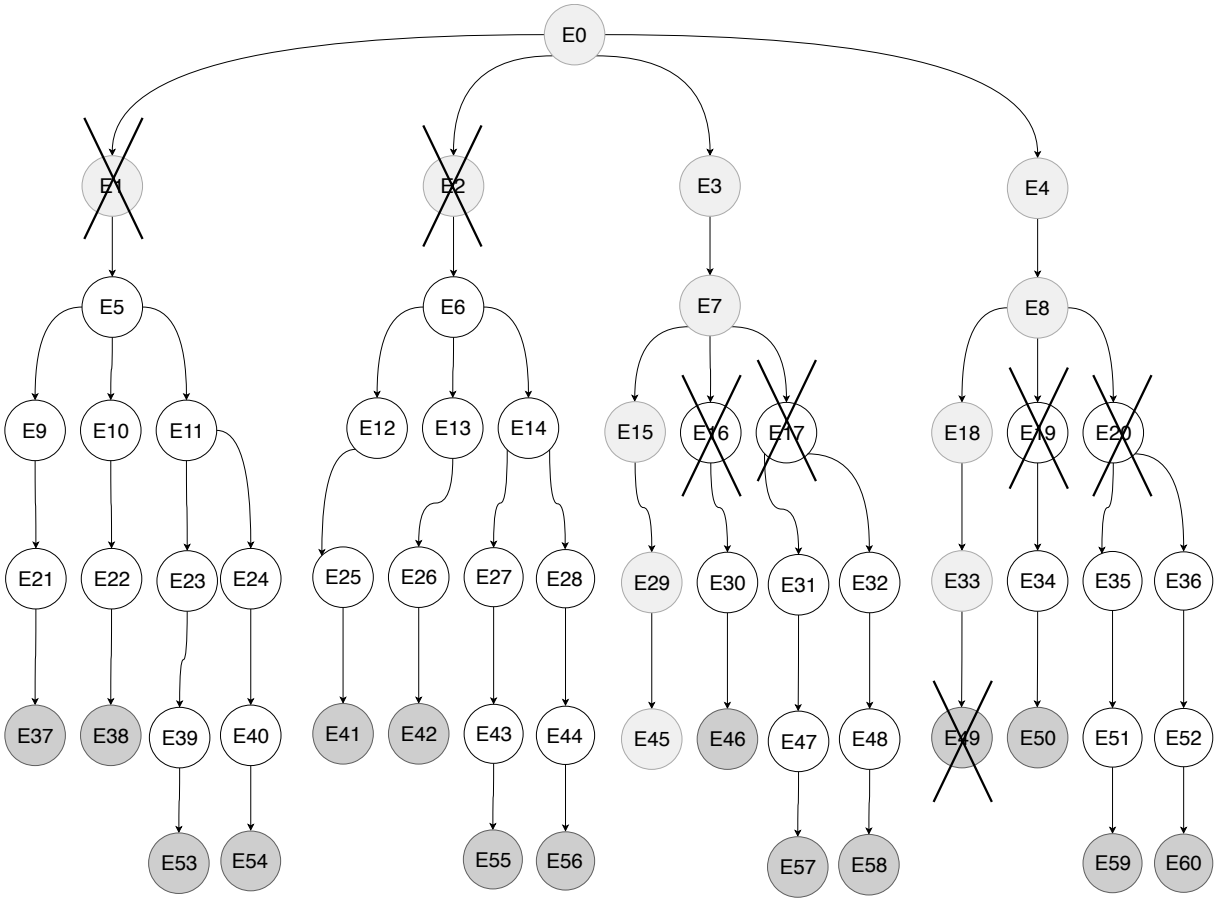


FIGURE 2.15 – Illustration du parcours de l'algorithme "CORA" des noeuds avec remaining

2.7.4 Mesure des performance des outils de model-checking

Pour pouvoir comparer des stratégies de model-checking, nous avions besoin d'un outil pour évaluer la rapidité de résolution et surtout l'espace mémoire consommé, qui est souvent le point critique en pratique. Nous avons choisi *memtime* que nous présentons dans cette section, car nous l'utiliserons pour les résultats présentés dans plusieurs des chapitres des parties II et III.

L'outil *memtime* [19] est un outil de mesure de la mémoire et du temps utilisé lors de l'exécution d'une commande. Il fonctionne actuellement sous Linux et Solaris-7. Il retourne certains paramètres comme $MaxRSS$, $MaxVSize$ et le temps d'exécution :

- $MaxRSS$ est un indicateur qui signifie Maximum Resident Set Size. Il indique la quantité maximale de mémoire d'un processus qui était dans la mémoire RAM. Il ne compte pas la mémoire qui a été échangée ou des parties de l'exécutable qui n'ont jamais été échangées. $MaxRSS$ est un indicateur des

contraintes pratiques pour résoudre le problème sur un ordinateur avec RAM limitée.

- $MaxVSize$ caractérise la quantité totale maximale de mémoire allouée au processus, ce qui inclut tous les types de mémoire, à la fois en RAM et en swap. $MaxVSize$ peut donner une image un peu erronée de la consommation de mémoire, car un processus peut allouer un espace d'adressage sans revendiquer la mémoire.

- Le temps d'exécution d'un processus est le temps passé par le système à exécuter le processus.

Memtime interroge ces valeurs à intervalles réguliers pendant l'exécution du programme et renvoie la valeur maximale à la fin du programme.

Lors de la comparaison des implémentations, $MaxRSS$ est généralement une meilleure mesure que $MaxVSize$, du moins si l'exemple est assez petit pour tenir dans la mémoire principale.

2.8 Conclusion

Dans ce chapitre, nous avons introduit le processus du model-checking et ses différentes phases, qui ont été décrites en détails chacune dans une section indépendante. Nous avons donné un aperçu sur les différents outils de model-checking et nous avons présenté en détails les outils utilisés dans le cadre de ce travail de thèse à savoir l'outil UppAal et l'outil UppAal-CORA. Un aperçu sur les quelques techniques rencontrées en littérature permettant de limiter l'explosion combinatoire qui représente la principale limite de Model-Checking a été fourni et l'outil qui permet de mesurer le temps et la mémoire allouée par un processus a été décrit.

A ce niveau, toutes les notions utiles pour la compréhension de nos travaux en modélisation et vérification pour l'agriculture de précision ont été introduites. Ces travaux ont été menés dans le contexte du projet AdAP2E. Les objectifs de ce projet sont présentés dans le chapitre suivant.

3 Adap2E : projet de robotique pour l'agriculture de précision

Sommaire

Comme déjà mentionné dans le chapitre 1, la robotique en agriculture a connu un grand progrès grâce à la révolution industrielle qui a conduit à une première phase de mécanisation de l'agriculture. L'objectif de la robotique en agriculture est de réduire la pénibilité des travaux agricoles et de garantir des niveaux de production suffisant face à l'accroissement de la population mondiale et à la difficulté d'exploitation des parcelles. Il est donc nécessaire de développer de nouveaux outils robotisés qui réduisent l'impact environnemental des activités humaines, qui assurent la sécurité des personnes et des opérateurs, et qui garantissent des niveaux de production suffisants pour nourrir la planète. Le projet Adap2E s'inscrit dans ce cadre, en proposant de développer des robots agiles capables d'accomplir plusieurs types de travaux dans des environnements agricoles, mais aussi capables d'interagir avec des personnes. Il faut noter que les travaux de thèse présentés dans ce mémoire s'inscrivent dans le cadre de ce projet qui les finance en partie.

3.1 Présentation du projet Adap2E

Le projet Adap2E ([90]) est un projet financé par l'ANR (Agence Nationale de la Recherche). Son objectif est de développer un démonstrateur robotique interactif et mécaniquement reconfigurable, capable d'effectuer des déplacements précis dans des environnements agricoles et naturels avec un degré d'autonomie ajustable à la situation rencontrée, et capable d'interagir avec les opérateurs et les personnes environnants.

Le milieu agricole est un environnement dynamique et incertain. Il est caractérisé par la diversité des tâches à réaliser et par la variabilité des conditions d'évolution (conditions météorologiques, géométrie du terrain, caractéristiques de sol, etc). Dans ce contexte dynamique, un robot reconfigurable doit être capable d'ajuster sa configuration mécanique et logicielle à la variabilité des situations. L'aspect reconfiguration est l'un des objectifs principaux du projet Adap2E. Le projet Adap2E s'articule autour de 4 axes :

1. Génération des lois de commande élémentaires et pilotage de la reconfiguration,
2. Planification, choix des modes de commande et supervision des lois associées,
3. Suivi de conception du démonstrateur et du simulateur,
4. Pilotage des équipes et du projet – Dissémination et recherche partenariale.

Nous avons été impliqué dans les actions de l'axe 2 de ce projet. Ce dernier, ainsi que notre mission, seront détaillés dans la section suivante.

3.2 Position des travaux de thèse dans le projet Adap2E

L'axe 2 concerne la planification, le choix des modes et la supervision des lois associées. Il se décline au travers des actions suivantes :

A) La définition des modes de commande,

B) La planification et la décision hors ligne,

C) L'étude de processus de supervision temps réel et de reconfiguration,

D) L'intégration des processus temps réel,

E) L'analyse des niveaux d'autonomie en fonction des modes de commande.

Nous intervenons plus précisément sur la partie B. Notre contribution avec cette thèse est la planification hors ligne, y compris pour le contrôleur du démonstrateur robotique créé à l'occasion du projet. Nous nous intéresserons aux aspects spatiaux comme temporels de cette planification : le robot doit pouvoir décider quelle action il effectue, à quel endroit, et à quel moment. D'après [105], la planification et la synthèse de contrôleur sont assez proches. Plus précisément, la planification consiste à trouver la séquence d'actions qui amènera le système dans un état désiré. La synthèse de contrôleurs consiste à trouver une exécution du système qui vérifie une

certaine propriété qui pourrait ne pas être assurée par le système. L'idée est d'utiliser la force de synthèse que le model-checking offre pour gérer la planification.

Nos travaux s'inscrivent parfaitement dans le contexte applicatif et environnemental du projet adap2E schématisé par la figure 3.1.

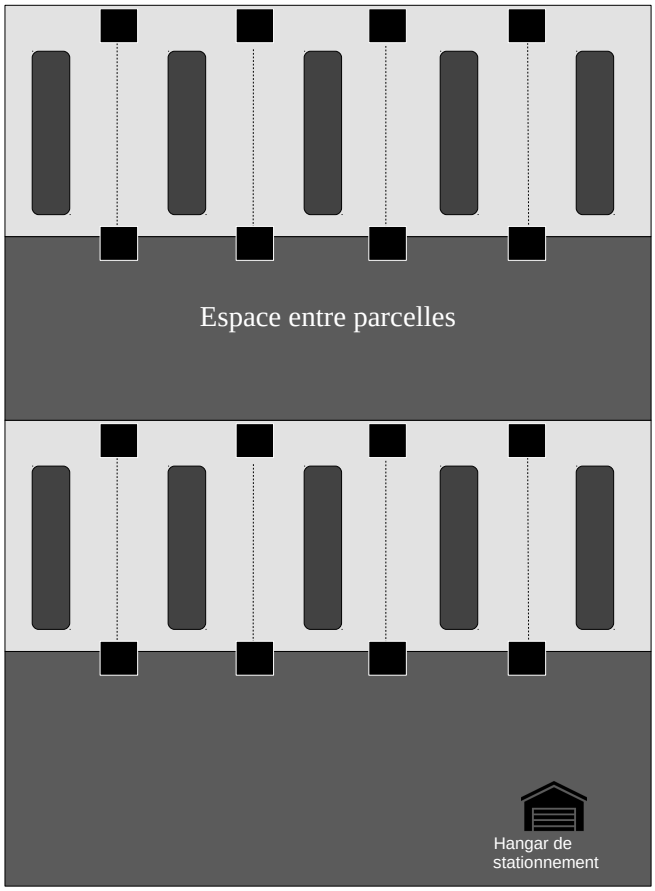


FIGURE 3.1 – Cadre du projet adap2E

Les carreaux noirs sont les points d'entrée ou de sortie de la parcelle, l'espace de couleur marron symbolise un environnement non structuré dans lequel le robot peut se déplacer librement, et l'espace de couleur verte représente un environnement structuré (par exemple, des parcelles de vignes plantées en rangs, les rangs étant représentés en vert foncé). Le robot se déplace entre ces environnements de différente nature. La position de son hangar de stationnement est connue avec précision. La mission du robot peut typiquement ressembler à la suivante : "Le robot part de son stationnement pour aller à la parcelle de vignes où plusieurs entrées sont possibles. Il réalise ensuite une opération agricole qui peut être, par exemple, de la pulvérisation ou de la récolte sélective dans les rangs de vigne. Lorsqu'il finit l'opération il peut

traiter une autre parcelle ou retourner à son hangar".
La mission du robot se déroule donc selon 4 phases principales exécutées séquentiellement :

1. Se rendre à une parcelle ;
2. Effectuer une opération agricole dans cette parcelle ;
3. Se déplacer entre deux parcelles ;
4. Rentrer au hangar.

Dans les phases 1, 3 et 4, le robot doit se déplacer entre 2 points dans un environnement non structuré, qui peut être partiellement connu à l'avance. Il est alors possible qu'il existe plusieurs chemins entre 2 points donnés. La question qui se pose est alors : "Lequel choisir ?". Il est également possible que le robot ait à choisir entre plusieurs points de départ et/ou d'arrivée. Ce choix dépendra potentiellement de l'opération agricole devant être effectuée dans la parcelle (phase 2). Le traitement de cette opération agricole doit être optimal pour un critère de coût par exemple la quantité de produit pulvérisé ou le temps de traitement. De plus, l'ordre du parcours des rangs n'est pas connu à l'avance. Par conséquent, on ne sait pas a priori quels sont les points d'entrée et de sortie, pour la parcelle 1 comme pour la parcelle 2.

Pour garantir l'optimalité de la mission et l'atteignabilté des parcelles par le robot, il faut vérifier si tous les chemins reliant les points d'entrées et de sorties sont éligibles. On parle d'atteignabilité de tous points à tous points, que nous appelons *atteignabilité totale bord à bord*. Cette question d'atteignabilité est l'objet du chapitre 6.

Dans la phase 2, le traitement de l'opération agricole doit être optimal. Cette problématique d'optimalité sera adressée dans les chapitre 4 et 5 de ce manuscrit. Le premier s'intéressera à l'optimisation d'une opération de pulvérisation sur un unique rang. Le second abordera le problème de l'optimisation d'une vendange sélective à travers une succession de rangs.

Pour toutes phases, il faut noter que le robot se déplace dans un environnement a priori connu. Il faut également remarquer que toutes les opérations que le robot peut effectuer au cours de sa mission dépendent de sa position, de son état interne et de son environnement. Deux objectifs peuvent alors être formulés :

1. Vérifier *a priori* que la mission est compatible avec les contraintes spatiales et la dynamique du robot,
2. Générer un contrôleur pour la gestion de la mission du robot.

Pour répondre à ces objectifs, nous avons choisi d'utiliser les techniques de model-checking. En effet, le model-checking permet de modéliser la dynamique du robot, de

vérifier la faisabilité de la mission dans le cas général et de générer le contrôleur (qui est optimal dans certains cas) pour la gestion de la mission. Notre but est de générer un contrôleur réalisable et optimal pour la réalisation d'une opération agricole.

3.3 Conclusion

Dans ce chapitre, le projet Adap2E a été introduit brièvement. Nous avons montré que l'ensemble des travaux du présent manuscrit s'inscrivait parfaitement dans le cadre applicatif et environnemental de ce projet. Les techniques de model-checking ont été choisies pour contribuer au développement méthodologique pour la vérification et la planification de travaux agricoles automatisés, avec parmi eux la pulvérisation en viticulture et la récolte sélective.

Dans la suite de ce document nous allons montrer que les techniques de model-checking peuvent se révéler très utiles pour aborder un questionnement de vérification de propriétés et d'optimalité qui peut être posé en agriculture de précision.

Deuxième partie

Application directe des techniques de model-checking à l'agriculture de précision

Introduction à la partie 2

Dans la partie I, nous avons introduit les fondamentaux de l'agriculture de précision et les bases du processus du model-checking. Toutes les notions dont nous avons besoin pour traiter des problèmes en agriculture de précision ont été introduites. Nous avons également vu que le projet de recherche Adap2E donne un cadre applicatif en robotique aux travaux qui seront présentés dans ce manuscrit.

Avant de présenter nos travaux sur l'apport du model-checking à l'agriculture de précision et à la robotique mobile, il nous reste à présenter un état de l'art sur les travaux antérieurs en model-checking appliqués à l'agriculture et à la gestion des éco-systèmes.

1 Résumé des applications du model-checking en agriculture dans la littérature antérieure

Les travaux de recherche publiés sur l'utilisation des techniques de model-checking en agriculture ne sont pas nombreux. En effet, nous n'avons identifié que trois thèses sur ce thème : les travaux de Christine Largouët ([84], [85], [83]), les travaux de Arnaud Helias ([70],[69], [74]) et les travaux de Yulong Zhao ([57], [86], [124]). Commençons par les travaux de Christine Largouët.

La thèse de Christine Largouët, intitulée "Aide à l'interprétation d'une séquence d'images par la modélisation de l'évolution du système observé : application à la reconnaissance de l'occupation du sol" [84] présente des méthodes permettant l'identification de l'occupation du sol à partir d'une série d'images aériennes et satellites d'une région agricole. Les images sont prises à différentes dates et chaque image contient environ 2000 parcelles [84]. L'objectif est de classer ces parcelles selon la culture présente. Les techniques traditionnelles de classification automatique d'images pouvaient correspondre plusieurs classes de plantation possibles à une même parcelle. L'approche de C. Largouët consiste à exploiter un modèle d'évolution de la parcelle et à utiliser la séquence d'images pour améliorer la classification [85]. Ce modèle, exprimé sous la forme d'un automate temporisé, représente l'évolution avec le temps des stades culturaux et de la succession des cultures. La séquence d'images permet de déterminer la dynamique d'évolution de la parcelle ce qui permet de mieux la classer. La démarche commence par une classification préliminaire des images qui

fournit l'ensemble de toutes les classes envisageables d'une parcelle. Une méthode de raffinement est ensuite utilisée pour affiner la classification. Cette méthode repose sur des mécanismes de prédiction et de postdiction, dans lesquels le modèle d'évolution de la parcelle est utilisé pour donner l'état attendu de la parcelle à chaque date. Une analyse d'atteignabilité par Model-checking permet d'affecter à chaque parcelle une seule classe. [83].

Dans [70, 69, 74], Arnaud Hélias a traité le problème de gestion des effluents d'élevage en utilisant le formalisme des automates temporisés. Le système de gestion des effluents est composé de deux types d'unité : les unités de productions (les élevages) et les unités de consommation (les cultures). Les élevages produisent de la matière organique peut être épandue sur les cultures à des fin de fertilisation. Le transfert de la matière organique entre les unités dépend de contraintes liées : à la nature de la matière organique, au type de transfert (inter ou intra exploitation), aux normes agronomiques et environnementales, pour le producteur aux périodes de disponibilité de la matière organique, pour le consommateur aux périodes du besoin de la matière organique. L'objectif de l'étude est d'assurer la gestion des effluents d'élevage en identifiant les possibilités de transfert de matière organique entre les unités respectant toutes ces contraintes. Arnaud Hélias a proposé une modélisation générique des contraintes temporelles de transfert entre les unités. Celles-ci ont été représentées individuellement sous forme d'automates temporisés. Arnaud Hélias a ensuite appliqué cette modélisation au problème de gestion des effluents d'élevage à la Réunion. Le model-checking a été utilisé pour simuler les décisions d'épandage de la façon suivante : une requête d'atteignabilité avec une fonction spéciale de Kronos permet de récupérer tous les chemins respectant la requête qui définit les contraintes nécessaires pour effectuer un transfert. A partir de ces chemins, la date au plus tôt et la date au plus tard du transfert sont déterminées. Une règle de décision permet de choisir l'instant précis pour le transfert, entre ces dates. Le model-checking est donc utilisé d'une façon particulière, à des fins de simulation d'un système.

Les derniers travaux antérieurs de thèse que nous allons présenter sont ceux de Yulong Zhao. Dans sa thèse [124], Zhao a traité deux problèmes : le premier concerne la gestion des écosystèmes [86] et le deuxième la gestion des pâturages [57].

L'écosystème considéré dans le premier problème est un réseau trophique de type proie-prédateur, comprenant quatre espèces de poisson : le thon, le maquereau, la sardine et l'anchois. Le thon est un prédateur du maquereau et le maquereau est un prédateur des sardines et des anchois. Deux pressions de pêche sont considérées : le thon et le maquereau. Une perturbation environnementale, le réchauffement climatique, est prise en compte. Zhao a proposé un modèle qualitatif de cet écosystème nommé EcoMata. 3 niveaux de biomasse sont considérés pour chaque espèce de poisson (*Low*, *Normal* et *High*). Le modèle est exprimé sous forme d'un ensemble d'automates temporisés où chaque espèce de poisson est représentée par un automate décrivant l'évolution entre chacun des 3 niveaux de biomasse. La pression halieutique est représentée (les forces de pêche) est représentée par des automates

décrivant les niveaux qualitatifs et les contraintes temporelles décrivant la durée de la pression. Le changement d'état qualitatif d'une espèce est déclenché par un événement de synchronisation émis par un automate en interaction avec cette espèce (proie, prédateur, pression de pêche et perturbation environnementale). Le model-checking a été utilisé pour la simulation qualitative et prédictive de l'écosystème, avec un objectif de politique de gestion de la ressource en poisson.

Le système de gestion du pâturage est basée sur la production d'herbe. Une production élevée d'herbe nécessite des fauchages et une production insuffisante d'herbe implique la consommation du fourrage conservé. Pour les exploitations ne disposant que d'une petite surface de prairies, l'utilisation d'une fertilisation azotée est courante pour stimuler la production d'herbe. Ainsi, les activités considérées pour la gestion du pâturage sont la mise au pâturage, le fauchage et la fertilisation. Zhao propose une approche originale de modélisation du pâturage : les activités liées au pâturage sont modélisées par des automates temporisés et la croissance de l'herbe est modélisée par un automate hybride. Le modèle hybride hiérarchique de pâturage est divisé en quatre couches : Prairie (pour modéliser de la dynamique propre de la croissance de l'herbe), Exécution (pour modéliser les activités du pâturage), Contrôleur (pour modéliser la prise de décision dans un système de pâturage) et Horloge centrale (pour modéliser l'avancement du temps dans le système du pâturage). Les techniques de model-checking ont été utilisées pour la recherche de stratégies optimales de mise au pâturage et de fertilisation avec l'outil UppAal-CORA.

Dans tous les travaux que nous venons de présenter, les techniques de model-checking ont été utilisées pour la simulation ou l'optimisation d'un système. Nos travaux, font aussi appel aux techniques de model-checking, avec des modèles spatialisés, sont, à notre connaissance, les premiers en agriculture de précision.

Comme déjà indiqué, les opérations agricoles sont des opérations très pénibles à effectuer et la main d'oeuvre qualifiée est devenue de plus en plus rare. On observe donc une automatisation croissante des opérations agricoles et la mise en œuvre de l'agriculture de précision repose sur des équipements agricoles. Nous nous intéressons ici aux systèmes automatisés mobiles pour l'agriculture que nous allons évoquer après avoir présenté le cadre des systèmes automatisés mobiles de façon plus générale.

2 Les systèmes automatisés mobiles

Un système automatisé mobile est un système qui effectue des actions, prend des décisions et agit en temps réel de manière autonome ou avec une intervention humaine limitée. Par exemple, le robot aspirateur est un système automatisé mobile capable de réaliser le travail d'un aspirateur de façon autonome, il est capable de réagir aux obstacles, de les contourner, de les mémoriser, de se charger de manière autonome de mettre à jour ses informations si par exemple un obstacle disparaît, etc

[120, 75]. Les voitures autonomes sont un autre exemple de système automatisé mobile. Les voitures autonomes font l'objet de développements scientifiques constants. Elles ne font pas l'objet de ces travaux de thèse mais une idée générale sur les avancées dans ce domaine ne peut être que pertinente dans le contexte de la robotique mobile autonome et de la robotique agricole. Nous avons choisi de citer ci-après trois articles. Le premier [24] présente une chronologie détaillée sur les voitures autonomes. Le deuxième [56] décrit une revue critique des technologies fonctionnelles nécessaires pour le développement et le déploiement des véhicules terrestres autonomes. Le dernier [106] présente la nécessité de prendre en compte les facteurs humains dans les voitures autonomes ; il définit un partenariat entre le conducteur et la voiture et la dynamique décisionnelle à prendre en compte.

La chronologie des voitures autonomes commence en 1926 avec la première voiture radio commandée au monde [24]. La technologie des voitures autonomes a beaucoup progressé grâce au développement des systèmes guidés par vision utilisant le LIDAR, le radar, le GPS et la vision par ordinateur. Un bilan chronologique détaillé des développements réalisés dans le domaine de l'automatisation est présenté dans le document [24]. D'après [56], cinq domaines technologiques fonctionnels ont été identifiés pour le développement des véhicules terrestres autonomes : la localisation, la navigation, la communication, la planification et la mobilité. L'auteur de [56] a établi comment ces cinq domaines technologiques s'intègrent comme un tout fonctionnel. Il considère que les capteurs, les algorithmes et les méthodes nécessaires pour le développement des véhicules terrestres autonomes existent, et estiment qu'ils sont viables au plan opérationnel et pourraient être déployés sous des formes spécifiques. Il resterait cependant impossible de développer une "machine autonome" générique pour toutes les missions et pour tous les terrains. L'auteur souligne la nécessité d'une définition complète et claire d'une mission.

Il est nécessaire de considérer l'interaction entre le système et son utilisateur. La présence d'un élément d'autonomie dans un système implique la délégation du pouvoir de contrôle au système [106] par l'utilisateur. On peut considérer plusieurs niveaux d'autonomie pour un système, depuis le niveau 1 dans lequel c'est l'utilisateur qui prend toutes les décisions et contrôle directement le système, jusqu'au le niveau 10 dans lequel le système qui prend toutes les décisions et ignore l'utilisateur. Entre ces deux niveaux extrêmes, il existe d'autres niveaux où une collaboration entre l'utilisateur et le système autonome s'établit. Dans ces niveaux, l'utilisateur joue un rôle de superviseur dans un système complexe qui offre des conseils à l'utilisateur. Dans [106] sont décrits les différents paradigmes/modèles d'interaction existants (le modèle traditionnel de définition des niveaux d'automatisation proposé par [112] en 1978 et révisé par [100] en 2000, le modèle PACT (Pilot Authorisation and Control of Tasks), utilisé dans le domaine aérospatial, définissant 3 modes d'automatisation : entièrement automatique, assisté et commande humaine et le modèle de classification NHTSA (National Highway Traffic Safety Administration), définissant plusieurs niveaux de conduite autonome). Un nouveau modèle est proposé dans

[106] pour mettre en évidence la relation entre l'utilisateur et le véhicule en termes de contrôle et de délégation de l'autorité. Pour que l'interaction entre l'utilisateur et le véhicule soit efficace, il est important de concevoir un système qui permette à l'utilisateur de comprendre non seulement ce que le système est en train de faire (et prévoit de faire), mais aussi ce qu'il ne peut pas faire. Cela crée un partenariat de confiance entre l'utilisateur et le système qui reconnaît non seulement les limites humaines mais les combine avec des limitations systémiques afin de déterminer un système centré sur l'utilisateur pour la conduite autonome.

Dans notre contexte, nous nous intéressons plus particulièrement aux systèmes automatisés mobiles pour l'agriculture.

3 Systèmes automatisés mobiles en agriculture

Les systèmes automatisés mobiles conçus pour l'agriculture de précision peuvent effectuer des actions qui varient dans l'espace et dans le temps en fonction des informations provenant de capteurs (capteur Lidar, capteur gps ...) ou d'une carte de mission. La recherche sur les équipements autonomes dans l'agriculture a débuté au début des années 1960, en se concentrant à l'époque sur le développement de systèmes de guidage automatique [75]. L'automatisation des robots agricoles est devenue aujourd'hui essentielle pour améliorer l'efficacité du travail et réduire la pénibilité du travail manuel [42]. Au cours de la dernière décennie, un nombre croissant de projets de recherche sur les robots agricoles autonomes ont été entrepris [72], [77], [114]. Les opérations agricoles sur les cultures agricoles sont très variées et elles peuvent être classées en plusieurs groupes. On peut citer par exemple, le semis, la fertilisation, l'irrigation, la pulvérisation, le désherbage ou la récolte. Chaque opération peut être effectuée sur une variété de culture et l'exécution d'une opération sur une culture donnée n'est pas nécessairement applicable de la même manière à une autre culture. Cette variété rend complexe le contrôle et l'automatisation des opérations agricoles. La complexité augmente lorsque l'on considère l'environnement et les cultures agricoles. En effet, l'environnement agricole n'est pas uniforme pour toute les cultures : c'est un environnement non structuré, caractérisé par une variabilité continue de la visibilité, de l'éclairage, des conditions atmosphériques, etc [11]. Les cultures agricoles tels que les vignes, l'arboriculture, les pommes de terres, etc, sont caractérisées par une grande variabilité de forme, de texture, de couleur, de taille, d'orientation et de position [11] [1]. Il est alors difficile, voire impossible de développer des méthodes génériques pour l'automatisation et le contrôle des opérations agricoles, applicables à toutes les cultures. Néanmoins, de nombreux travaux

1. D'après [11], le monde robotique peut être divisé en quatre groupes en fonction des caractéristiques structurelles des environnements et des objets : 1) l'environnement et les objets sont structurés ; 2) l'environnement n'est pas structuré et les objets sont structurés ; 3) l'environnement est structuré et les objets non structurés, et 4) l'environnement et les objets sont non structurés. Le domaine agricole est associé au quatrième groupe dans lequel rien n'est structuré.

de recherche se sont intéressés à la robotique agricole. Ces travaux sont spécifiques pour chaque opération et pour chaque culture. Le tableau ci-dessous en présente des exemples.

Opération agricole	**Cultures et références**
Désherbage	riz [42], tomates [89]
Fertilisation	blé [119], Canne à sucre [50]
Pulvérisation	vigne [62, 61], pèches et abricots [61]
Récolte	pommes, oranges et raisins [28], melon [126]

TABLE 3.1 – Exemples d'opération agricole sur certaines cultures

La section suivante présente les opérations agricoles qui ont servis de base applicative à nos travaux, et qui seront traitées dans la suite de ce manuscrit.

4 Les opérations agricoles traitées dans ce manuscrit

Comme nous l'avons déjà mentionné, l'objectif général de ces travaux de thèse est d'explorer les capacités des méthodes formelles et en particulier les techniques de model-checking à résoudre des problèmes issus de l'agriculture de précision et leur mise en oeuvre avec des systèmes automatisés mobiles. Dans ce mémoire, trois problèmes agricoles sont traités : l'optimisation locale de la pulvérisation dans un rang de vigne, l'optimisation de la récolte sélective sur un champ de vigne, et la vérification d'une mission dans un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques. Ces trois problèmes visent à déterminer comment mettre en oeuvre une machine agricole automatisée pour respecter deux types de contraintes : 1) celles liées à la dynamique de la machine agricole ; et 2) celles issues de l'agriculture de précision, comme par exemple pulvériser suffisamment pour protéger la plante des maladies tout en ne pulvérisant pas trop pour protéger l'environnement. Tous ces problèmes ont en commun la gestion de la composante spatiale, qui se retrouve selon les trois représentations complémentaires que nous avons introduit dans la section 3 : un environnement structuré en rangs traité avec une approche locale "1D", c'est-à-dire rang par rang, un environnement structuré en rangs traité au niveau "1D+", c'est-à-dire avec des critères considérés au niveau global sur la totalité des rangs, et enfin un environnement ouvert "2D" permettant des déplacements plus libres. Tous ces exemples sont issus d'une analyse de situations concrètes.

Pour résoudre ces problèmes, nous allons appliquer les méthodes formelles basées sur le model-checking. Dans les cas d'optimisation (pulvérisation, récolte), il s'agit, au moyen du model-checking et d'un algorithme CORA, de calculer un contrôleur optimal. Dans le cas de la vérification de la mission agricole, il s'agit de vérifier

qu'il est possible de déterminer un contrôleur réalisant les objectifs de cette mission. Pour chacun de ces problèmes, nous avons établi une méthode spécifique. Chacune de ces méthodes s'appuie sur le processus de model-checking qui nécessite dans un premier temps une modélisation formelle du comportement du système, puis une représentation des propriétés à vérifier sur le système pour sa validation, pour enfin effectuer le processus de vérification des propriétés par model checking, et l'optimisation simultanée le cas échéant. Chacune de ces méthodes utilise les outils présentés dans le chapitre 2, c'est-à-dire les formalismes de modélisation des automates temporisés et leur extension les automates temporisés de coût, la logique temporelle TCTL pour l'expression des propriétés, et les outils de model-checking UppAal et UppAal-CORA.

4 Problème de pulvérisation

Sommaire

4.1 Introduction

L'agriculture de précision a pour origine le développement de technologies de localisation et de détection par lesquelles une gestion spécifique au site peut être mise en œuvre sur le terrain, afin d'optimiser un critère donné, par exemple pour en tirer un bénéfice économique ou environnemental [23]. Les technologies les plus récentes, telles que l'imagerie multi-spectrale et le balayage laser, qu'elles soient aéroportées ou au sol, permettent l'acquisition de données de culture avec une grande résolution spatiale ([108], [68]). Par exemple sur les cultures pérennes telles que la vigne, des capteurs peuvent être utilisés pour quantifier et cartographier la densité de la végétation dans la canopée. Cette information qualitative et spatiale est l'occasion pour les spécialistes de l'automatisation de développer des méthodes innovantes et efficaces pour une pulvérisation précise des produits phytosanitaires. Celle-ci peut contribuer à réduire les quantités de produits chimiques pulvérisés tout en maintenant le rendement et la qualité.

Les produits phytosanitaires visent à protéger les plantes des parasites, dans les domaines agricoles et horticoles. Dans les exploitations agricoles, ils sont pulvérisés depuis des pulvérisateurs portés ou remorqués et parfois depuis un pulvérisateur aéroporté, intégré à un avion ou un hélicoptère. Les pulvérisateurs aéroportés sont souvent utilisés aux États-Unis [79] mais ils sont interdits en Europe par la directive n° 2009/128/CE [1]. La France est le troisième utilisateur mondial de produits phytosanitaires après les Etats-Unis et le Japon et le premier utilisateur européen [101]. En France, l'utilisation excessive de pesticides a entraîné la contamination du sol et la pollution de l'air. Or d'après [113], les quantités de produit perdu pendant la pulvérisation, relativement aux quantités émises sont comprises entre 10 et 45% sur le sol et entre 15 et 40% dans l'air. L'article [101] précise les risques liés à l'utilisation des pesticides sur la santé humaine et l'environnement. Les effets néfastes observés ont provoqué une prise de conscience mondiale de la nécessité de développer une agriculture protectrice des ressources naturelles. En 2008, le gouvernement français a adopté un plan ambitieux de politique environnementale, «Ecophyto 2018», visant à réduire de moitié l'utilisation de pesticides dans les 10 années suivantes. Mais entre 2008 et 2013, il n'a pu être démontré de baisse des ventes de pesticides [73]. La version 2 du plan Ecophyto a été lancée en 2015 avec un objectif de réduction de 50% reporté à 2025. La réduction des quantités de produits phytosanitaires est devenue une priorité dans les politiques Santé-Environnement et les conditions d'utilisation de ces produits sont devenues de plus en plus strictes vue leur impacts sur l'environ-

1. La pulvérisation aérienne de pesticides est susceptible d'avoir des effets néfastes importants sur la santé humaine et l'environnement, à cause notamment de la dérive des produits pulvérisés. Il convient donc d'interdire d'une manière générale la pulvérisation aérienne, avec possibilité de dérogation seulement lorsque cette méthode présente des avantages manifestes, du point de vue de son incidence limitée sur la santé et sur l'environnement par rapport aux autres méthodes de pulvérisation, ou lorsqu'il n'existe pas d'autre solution viable, pourvu qu'il soit fait usage de la meilleure technologie disponible pour limiter la dérive.

nement et la santé humaine. C'est dans ce contexte que s'inscrit notre contribution qui propose une méthode permettant d'optimiser le procédé de pulvérisation en milieu agricole.

Dans cette partie du document, nous proposons une contribution au développement des méthodes de commande des pulvérisateurs. Notre objectif est d'optimiser la pulvérisation en fonction de la végétation, dans le cas des cultures palissées telles que la vigne. En particulier, compte tenu de la dynamique de contrôle d'un pulvérisateur, nous déterminons une carte spatiale des commandes de pulvérisation à appliquer. Les commandes de pulvérisation diffèrent entre elles par la dose de pesticides à pulvériser et par la configuration d'usage des organes de pulvérisation. Notre contribution, appelée Modélisation par Automates pour la Pulvérisation de Précision, en anglais Automata Modelling for Precision Spraying (AMPS), est basée sur l'outil de vérification UppAal-CORA. Comme indiqué dans le chapitre 2, UppAal-CORA [16] est une variante de UppAal qui permet l'analyse d'accessibilité à coût optimal pour les automates temporisés de coût. Il a été utilisé avec succès pour des problèmes de planification et de routage dans plusieurs études de cas ([16] : planification de tâches, problème d'atterrissage d'aéronef, problème de routage de véhicule avec fenêtres temporelles, etc.). A notre connaissance, l'utilisation que nous avons faite de ces techniques de model-checking et d'analyse d'accessibilité à coût optimale est novatrice dans le domaine de la pulvérisation de précision et dans le domaine plus large de l'agriculture de précision.

Plusieurs auteurs tels que [62] et [99] ont développé des méthodes d'automatisation pour les pulvérisateurs. L'évaluation des performances est généralement empirique : une fois le contrôle conçu et mis en œuvre, il est testé sur un système existant puis les performances sont analysées. Au contraire, la méthode AMPS vise à estimer les performances à atteindre avant de développer le pulvérisateur automatisé. Notre étude suppose que l'on utilise un pulvérisateur pneumatique équipé d'un contrôle individuel des organes de pulvérisation. Mais la méthode AMPS a été développée pour offrir une adaptabilité à d'autres technologies de pulvérisation. Elle prend en compte deux critères que l'on cherchera à satisfaire. L'un, vise à garantir localement une protection phytosanitaire suffisante sur chaque plant de vigne. L'autre, vise à réduire globalement la quantité de pesticides épandue à l'échelle de la parcelle.

Le plan de ce chapitre est le suivant. La section 2 décrit le matériel expérimental ainsi que les caractéristiques du pulvérisateur choisi. La section 3 présente la méthode de modélisation des automates pour la pulvérisation de précision. Enfin, dans la section 4, la méthode AMPS est appliquée aux données LiDAR (Light Detection And Ranging) d'une vigne réelle.

4.2 Description du système matériel

Dans cette section, le système matériel, sur lequel la méthode AMPS a été déployée est décrit. Dans la première sous-section, nous rappelons les différents types de pulvérisateurs existants et la technologie que nous avons retenue. Dans la seconde, le pulvérisateur que nous avons imaginé est décrit, ainsi que ses caractéristiques. Enfin, le système LiDAR qui permet l'acquisition des données qui seront par la suite traitées par la méthode AMPS est présenté.

4.2.1 Les pulvérisateurs pneumatiques classiques

Il existe plusieurs types de pulvérisateurs : rampes premiers traitements, aéroconvecteurs, voûtes pneumatiques, face par face à jet porté ou jet projeté, face par face pneumatiques, panneaux récupérateurs à jet porté. Pour la mise au point de notre méthode, nous nous sommes basés sur l'exemple du pulvérisateur Eco semi-porté du constructeur Calvet (voir figure 4.1), qui est de type voûte pneumatique. Ce pulvérisateur dispose de 4 organes de pulvérisation : une main basse (LH), une main haute (HH), un canon haut (CH) et un canon bas (CB). Une "main" est un organe de pulvérisation comprenant 2 ou 3 buses, qui vont fonctionner en même temps. Lorsque le pulvérisateur est utilisé en mode "passage un inter-rang sur 4", les mains pulvérisent sur les rangs de vigne adjacents au pulvérisateur et les canons pulvérisent sur les rangs de vigne distants. Dans la suite, pour simplifier la modélisation et les descriptions, on ne s'intéressera qu'aux mains du pulvérisateur.

FIGURE 4.1 – Exemple du pulvérisateur Eco semi-porté du constructeur Calvet

Dans notre modélisation décrite ci-après, nous appellerons "buse" un organe de pulvérisation, même s'il comprend, en fait, plusieurs buses fonctionnant simultanément. En principe, les pulvérisateurs pneumatiques que l'on trouve sur le marché ne comportent pas d'automatismes permettant de fermer ou ouvrir des organes de pulvérisation de façon indépendante en temps réel. La commutation de ces organes se fait par réglages manuels. Pour notre pulvérisateur, par exemple, les deux mains

sont habituellement utilisées en même temps. Cependant, on peut trouver des prototypes automatisés ([62, 122]), et cette automatisation est un des outils importants pour la pulvérisation de précision. C'est pourquoi nous avons pris en compte cette particularité dans notre méthode, et donc dans notre modélisation. Le modèle a été réalisé pour pouvoir être adapté à des automatismes pouvant relever d'autres technologies de pulvérisation, par exemple, des appareils à jet porté.

4.2.2 Pulvérisateur ciblé

Pour aborder la pulvérisation de précision, nous supposons que notre pulvérisateur peut activer/désactiver ses mains de manière indépendante et possède, en plus de ses mains haute (HH) et basse (LH), une troisième main centrale (CH), qui peut couvrir approximativement la même hauteur de culture que LH et HH utilisées ensemble (figure 4.2). La pulvérisation avec la main centrale seule ne permet cependant pas de couvrir la même densité que les deux mains HH et LH. En effet, chaque main a le même débit, soit la moitié du débit nominal (le débit nominal est celui que l'on obtient en utilisant simultanément LH et HH, soit le mode "par défaut" en pleine végétation). La position verticale de la main sur le pulvérisateur et l'écart latéral entre la main et la végétation définissent la hauteur de végétation couverte par la main.

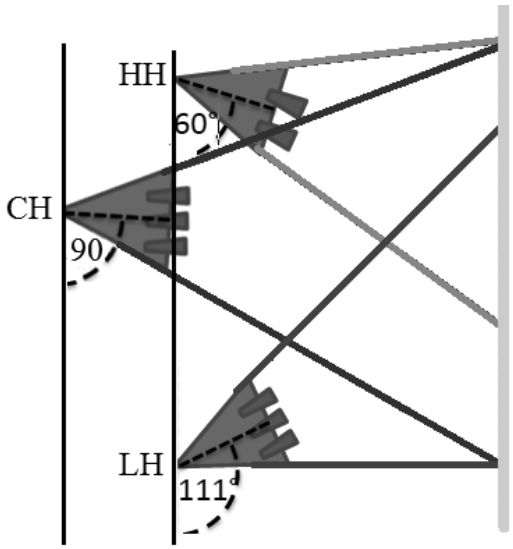


FIGURE 4.2 – Disposition des mains du pulvérisateur ciblé

4.2.3 Description du système LiDAR utilisé

Cette sous-section décrit le système LiDAR (Light Detection And Ranging) utilisé. Il permet de réaliser un "scan" de la végétation à proximité du pulvérisateur. L'analyse des données recueillies permettront de caractériser la densité de végétation et d'adapter en conséquence la configuration de pulvérisation.

Les données LiDAR terrain que nous avons utilisées dans notre étude sont issues d'un balayage planaire des rangs de vigne réalisé par un LiDAR 2D de type LMS 100 [2] monté sur un tracteur tel que présenté dans [10]. La vitesse du tracteur était de 5 $km.h^{-1}$. La fréquence d'acquisition des données était de 50 Hz. Cela permet un scan de toute la hauteur de végétation tous les 3 cm. La plage angulaire de balayage étant de 270 ° avec une résolution de $0,5$ °, un scan est donc constitué de 541 points. L'appareil était orienté de manière à balayer la végétation des deux côtés (donc 2 rangs à la fois) ainsi que le sol.

4.3 Méthode AMPS

Cette section présente la méthode AMPS (Automata Modeling for Precision Spraying) que nous avons développée afin d'aborder les problèmes d'agriculture de précision. AMPS prend en entrée les données LiDAR 2D de la canopée. Elle calcule, pour un pulvérisateur donné avec contrôle individuel des mains, une séquence de commande optimisée pour un critère de coût, tout en assurant une protection suffisante sur chaque plant de vigne par des contraintes assurant localement une dose minimale de pesticides répandue à chaque instant. Le débit et le temps de réponse pour l'ouverture ou la fermeture des buses sont des paramètres pris en compte dans la modélisation car ils vont jouer sur le choix de la commande la plus appropriée pour le pulvérisateur.

La méthode AMPS se décompose en 3 étapes, représentée sur la figure 4.3 :

Etape 1 : Elle permet de définir une fonction de correspondance entre les états de densité de la végétation et les configurations de pulvérisation souhaitées pour cet état. Cette fonction, appelée *Precision Spraying Mapping* (PSM), associe à chaque état de végétation possible deux configurations de pulvérisation souhaitées pour cet état : la configuration la plus adaptée C_{best}, et une configuration alternative C_{alt}. Ces deux configurations garantissent une protection suffisante pour chaque plant de vigne.

Etape 2 : Application d'un algorithme de traitement de données sur des données LiDAR brutes issues d'un scan d'un rang de vigne. Cet algorithme utilise la fonction PSM pour produire une carte de préconisation spatialisée pour la

2. fiche produit : `https://cdn.sick.com/media/pdf/1/41/841/dataSheet_LMS100-10000_1041113_fr.pdf`

pulvérisation du rang de vigne et des règles d'agrégation et de filtrage pour former des blocs de végétation dont la durée de traitement avec le pulvérisateur est cohérente avec les temps de réponse des buses de pulvérisation. Cette étape consiste donc à :

- utiliser des règles d'agrégation et de filtrage pour former des blocs de végétation de densité homogène dont la longueur peut varier d'un bloc à l'autre,
- déterminer pour chaque bloc identifié les configurations de pulvérisation (C_{best} et C_{alt}) adaptées à la densité de végétation de ce bloc.

Etape 3 : Permet de produire une carte de commandes du pulvérisateur, avec, pour chaque bloc de végétation identifié, sa durée de traitement et la configuration de pulvérisation retenue. La carte spatiale obtenue à l'étape 2 est une donnée d'entrée du modèle PTA_AMPS présenté en section 4.3.3. PTA_AMPS modélise l'ensemble de l'opération de pulvérisation et la dynamique de l'équipement utilisé. L'exploitation de ce modèle à l'aide de l'outil UppAal-CORA permet alors de calculer la séquence de commande optimale qui permet de pulvériser un rang avec une quantité globale de produit minimale. Pour chaque bloc, la configuration retenue est stockée dans C_{opt}.

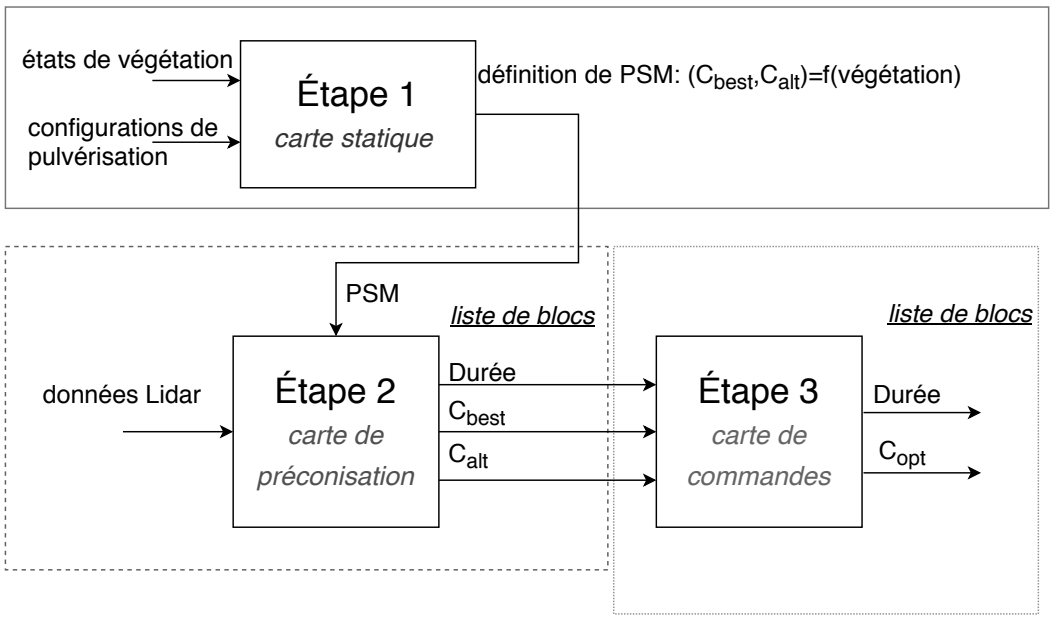


FIGURE 4.3 – La méthode AMPS

Dans ce qui suit, chacune de ces étapes est décrite plus en détails.

4.3.1 Étape 1 : Définition de la fonction de correspondance

La première étape de la méthode AMPS a pour objectif principal de construire la carte statique (PSM) de pulvérisation. Pour ce faire, elle prend en entrées des valeurs qualitatives dérivées des données LiDAR permettant de caractériser la densité spatiale de la végétation, et les différentes configurations possibles du pulvérisateur. Elle génère en sortie, un ensemble de configurations de pulvérisation envisageables pour chaque état de végétation possible.

4.3.1.1 Caractérisation de la végétation

Un champ de vigne est organisé en rangs. Nous cherchons ici à décomposer chaque rang en une suite de blocs de densité de végétation homogène, pouvant être de différentes longueurs. Pour identifier ces blocs de densité de végétation homogène, nous décomposons la canopée en 3 sections horizontales (découpées selon l'axe vertical). Après calcul des blocs de végétation, chacun d'eux sera considéré comme homogène en densité. La hauteur d'un bloc de végétation est donc divisée en 3 zones (voir figure 4.4) :

- La section "Low" (L) : pour la hauteur de la végétation comprise entre h1 et h2 ;
- La section "Middle" (M) : pour la hauteur de la végétation entre h2 et h3 ;
- La section "High" (H) : pour la hauteur de la végétation supérieure à h3 et inférieure à une hauteur maximale définie.

On considère également la hauteur totale comme une section T ("Totale") telle que la hauteur de T est la somme des hauteurs de L, M et H. La zone inférieure à h1 n'est pas considérée parce qu'il y a souvent de l'herbe dans cette zone basse, qui ne doit pas être prise en compte pour la pulvérisation de la vigne. Pour notre exemple, h1, h2 et h3 ont été réglés respectivement à 0.3, 0.6 et 1.2 m pour une hauteur de vigne maximale de 2 m.

Pour une section donnée (T, L, M, H), 3 valeurs de densité peuvent être considérées :

- 0 est la valeur représentant l'absence de végétation,
- 1 représente un peu de végétation,
- 2 représente beaucoup de végétation.

Ces valeurs sont obtenues à partir des données LiDAR 2D brutes en fonction de seuils déterminés expérimentalement. Le nombre d'interceptions de faisceaux LiDAR dans chaque section à l'intérieur d'un bloc est supposé représentatif de la quantité de feuillage et de la porosité de la canopée. En effet, si le nombre de

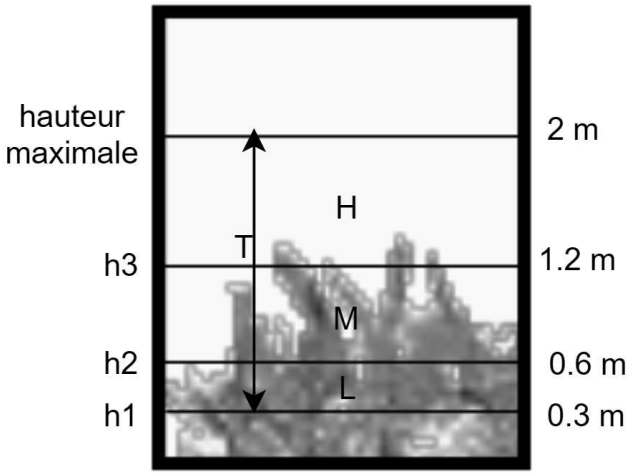


FIGURE 4.4 – Illustration de la division de la hauteur de végétation

points d'interceptions est élevé alors la porosité est faible et inversement. Dans cette contribution, les seuils ont été calculés de façon empirique en se référant à des statistiques effectuées sur ce seul rang de vigne, pour chaque section horizontale (T, L, M, H). Le seuil de discrimination entre 1 et 2 a été choisi comme étant la médiane du nombre d'interceptions de faisceaux LiDAR dans la section horizontale considérée.

Les seuils pour la valeur abstraite 0 ont été choisis spécifiquement pour chaque section horizontale :

- section L : exactement 0 interception
- section M : moins de 30 interceptions LiDAR pour une tranche de 10 cm de large
- section H : moins de 15 interceptions LiDAR pour une tranche de 10 cm de large

Il faut noter que pour la section T seules les valeurs 1 et 2 ont été considérées. La décision de considérer un bloc de végétation comme «végétation manquante» ou «trous» est prise à partir de la section M. En effet, les statistiques montrent que pour tous les blocs de végétation, si la section M d'un bloc a "0" comme valeur d'état de végétation alors la section H du même bloc a aussi la valeur "0", ce qui démontre qu'il s'agit vraiment d'un trou. L'attribution de la valeur qualitative "végétation manquante" ou "Trous" à une tranche a donc été défini par le critère T-H-M-L = xx0x.

4.3.1.2 Correspondance entre états de végétation et configurations de pulvérisation

La fonction PSM fait correspondre les configurations de pulvérisation à chaque valeur du quadruplet (T, H, M, L) du champ étudié. Cette cartographie doit être définie en utilisant des connaissances d'experts en pulvérisation. Dans cette étude, la fonction PSM a été produite en utilisant une expertise basée sur le test de pulvérisateurs réels sur un banc de vigne artificiel [98]. 7 configurations de pulvérisation ont été considérées. Leur nom est basé sur les mains utilisées pour la pulvérisation :

- 3 configurations avec une seule main, et donc un débit de 50% du débit nominal : LH, CH et HH.
- 3 configurations avec deux mains, soit un débit nominal : $LH\&CH$, $LH\&HH$, $CH\&HH$.
- la configuration toutes mains éteintes est notée -.

Pour faire correspondre ces configurations avec les états de végétation, commençons par décrire, en cohérence avec la figure 4.2, comment chaque main couvre les 3 sections H, M et L. Ceci est fait dans le tableau 4.1. Dans ce tableau, le code 2 signifie que la section considérée est "bien couverte" (en quantité), le code 1 que la section considérée est "juste couverte" et le code 0 qu'elle n'est pas couverte. La main du haut HH permet de bien couvrir le haut et un peu le milieu, elle sera utilisée lorsque l'étage supérieur de la vigne est dense et l'étage inférieur est sans végétation (`H-M-L=2-1-0`). La main CH permettant de bien couvrir le milieu, un peu le haut et un peu le bas de la vigne, elle est bien adaptée à une répartition végétale concentrée au milieu (`H-M-L=1-2-1`). Et enfin, la main LH permettant de bien couvrir le bas et un peu le milieu de la canopée, elle est dédiée au traitement d'une végétation basse (`H-M-L=0-1-2`).

Main de pulvérisateur	H	M	L
LH	0	1	2
CH	1	2	1
HH	2	1	0

TABLE 4.1 – Taux de couverture des mains de pulvérisateur sur les sections H, M et L

Si on a un état de végétation `H-M-L=1-1-0`, la configuration HH sera considérée comme la plus adaptée, car elle protège suffisamment la végétation tout en pulvérisant la dose minimale (demi dose). Le même raisonnement vaut pour les configurations CH et LH, avec respectivement les états de végétation `H-M-L=1-1-1` et `H-M-L=0-1-1`.

Pour l'utilisation des configurations avec 2 mains activées, on considère la couverture du feuillage par chacune des mains comme additive. Ainsi, en considérant que sur l'échelle qualitative $\{0, 1, 2\}$ on a $1 + 1 = 2$, et $2 + x = 2$, la configuration classique $LH\&HH$ permet de bien couvrir une végétation de type `H-M-L=2-2-2` (dense dans les 3 sections et sur toute la hauteur).

La fonction de correspondance PSM prend également en compte la densité totale de végétation (section `T`) en plus du triplet `H-M-L`. On recherche un débit nominal (2 mains activées) lorsque `T` vaut 2 et un débit à 50% lorsque `T` vaut 1.

Dans la Table 4.2, la fonction PSM est représentée pour 24 états de végétation considérés suivant le quadruplet `T-H-M-L`. Si M = 0 (voir encadré jaune dans la Table 4.2) alors le bloc de végétation est considéré comme «végétation manquante» ou «trous». Un dénombrement complet comprendrait 54 combinaisons possibles de `T-H-M-L` (2x3x3x3), mais certaines combinaisons ne correspondent pas à des états de végétation possibles et ne sont donc pas représentés dans la table. Par exemple, le cas `T-H-M-L=2-0-1-0` n'est pas possible. En effet, si T vaut 2 alors par définition cela signifie que le nombre d'interceptions dans la section T est supérieur à la médiane du nombre d'interceptions sur la totalité des sections. Or si L, M et H sont <2, cela signifie que le nombre d'interceptions dans chacune de ces sections est inférieur à la médiane dans la section considérée. La somme des interceptions sur l'ensemble devrait donc être inférieure à la médiane pour l'ensemble, soit une valeur de T inférieure à 2. Ce cas d'état de végétation n'est donc pas possible.

La table 4.2 présente les deux configurations possibles pour un état de végétation : la meilleure, et une autre considérée comme acceptable. Par exemple, si on considère un bloc de végétation qui a les caractéristiques `T-H-M-L=1-0-1-0` (2e ligne cases oranges), alors C_{best} = LH et C_{alt} = HH. La table 4.1 (et la figure 4.2) montrent que ces deux commandes suffisent à assurer une protection suffisante pour la zone M. La commande CH délivre la même dose mais mal répartie, avec des produits pesticides gaspillés à la fois dans la zone basse et la zone haute de la végétation. Elle doit donc être évitée. Le choix entre LH et HH a été fait d'après données expérimentales : les résultats des tests effectués sur le banc artificiel indiquent que LH est la meilleure commande, et HH une alternative acceptable.

4.3.1.3 Intérêt des commandes alternatives

La Table 4.2 assigne à chaque bloc de végétation deux commandes différentes pouvant être utilisées comme configurations de pulvérisation, toutes les deux garantissant une protection efficace des cultures :

- La commande préférée (C_{best}) pulvérise la meilleure dose (ni trop élevée ni trop basse) et correspond donc à la configuration de pulvérisation la plus appropriée.
- La commande alternative (C_{alt}) pulvérise une dose locale garantissant la pro-

T-H-M-L	C_{best}	C_{alt}
xx0x	–	–
1010	LH	HH
1011	LH	CH
1012	LH	LH
1021	LH	CH
1022	LH&CH	LH&CH
1110	HH	CH
1111	CH	CH
1112	LH&CH	CH
1121	CH	LH&HH
1122	CH&HH	LH&CH
1211	CH	CH&HH
1212	LH&HH	LH&CH
1221	CH	CH&HH
2012	LH	LH&CH
2021	LH&CH	LH&HH
2022	LH&CH	LH&HH
2112	LH&CH	LH&CH
2121	LH&HH	LH&HH
2122	LH&CH	LH&CH
2211	CH&HH	CH&HH
2212	LH&HH	LH&CH
2221	CH&HH	CH&HH
2222	LH&HH	LH&HH

TABLE 4.2 – Fonction de correspondance PSM (extrait de 24 combinaisons)

tection locale de la vigne mais qui peut être un peu plus élevée que la dose nécessaire.

Quand aucune alternative satisfaisante ne peut être définie, on considère alors que $C_{alt} = C_{best}$.

Le besoin d'avoir deux commandes au lieu de seulement C_{best} résulte du temps de réponse des buses. Le choix d'une commande de configuration de pulvérisation locale et idéale unique pour chaque bloc peut ne pas fournir la pulvérisation optimale sur l'ensemble du champ en raison des temps de transition induits par l'ouverture ou la fermeture des buses. En effet, lorsque la configuration de pulvérisation est modifiée, *toute buse à activer doit l'être avant l'instant où elle doit pulvériser, et toute buse à désactiver doit l'être après.* Cette règle permet d'éviter une pulvérisation localement insuffisante. De plus, pendant la phase de fermeture et d'ouverture d'une main, le

débit de la main concernée n'est pas nul. Donc une modification trop fréquente de la configuration de pulvérisation peut conduire globalement, au niveau d'une parcelle, à une surexposition inutile de la végétation aux pesticides. C'est précisément à cause de cet effet induit par la dynamique des buses qu'un traitement informatique est nécessaire de façon à optimiser le contrôle des buses tout en garantissant un traitement efficace de la végétation.

Considérons par exemple un rang de vigne composé de 3 blocs b_1, b_2 et b_3 avec comme états de végétation respectifs `T-H-M-L=1-0-2-2`, `T-H-M-L=1-1-1-0` et `T-H-M-L=1-1-1-1`. D'après la Table 4.2 (cases vertes), seules 2 séquences de commandes sont possibles : $S1 = (LH\&CH, HH, CH)$ et $S2 = (LH\&CH, CH, CH)$, présentées sur la figure 4.5. La première correspond aux commandes C_{best} pour l'ensemble des blocs. La seconde choisit la commande C_{alt} pour le bloc b_2 (tout en maintenant C_{best} pour les blocs b_1 et b_3 puisque pour ces derniers $C_{best} = C_{alt}$).

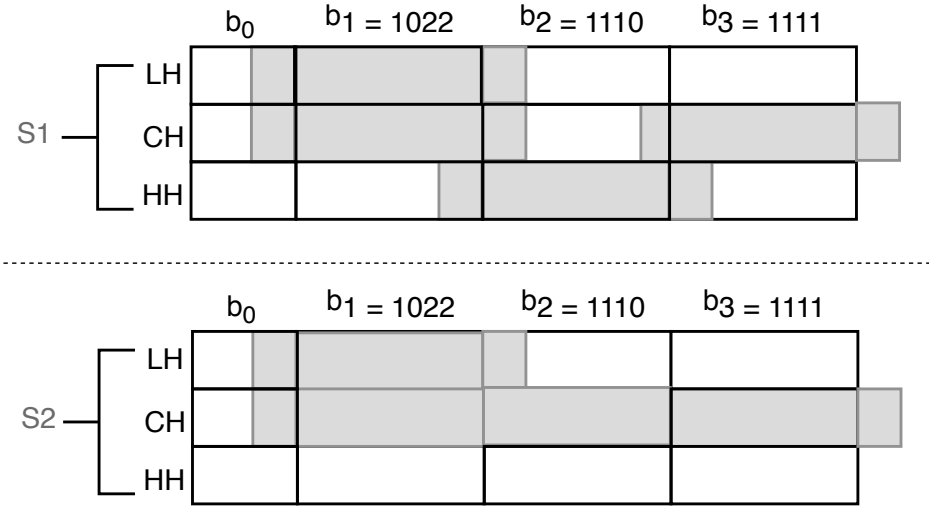


FIGURE 4.5 – Intérêt des commandes alternatives

Dans la figure 4.5, pour chaque buse, si une zone est verte alors la buse est active et si une zone est blanche alors la buse est fermée. Comme on peut le voir, le changement de commande pour le bloc b_2 dans la séquence $S1$ entraîne beaucoup plus de fermeture/ouverture des buses que pour la séquence $S2$. Si on fait le bilan noté CO du produit consommé pour chacune des 2 séquences, on a :

$$
\begin{aligned}
CO_{S1} &= Ouv(LH) + Ouv(CH) \\
&\quad + b1(LH) + b1(CH) + \textbf{ Ouv(HH)} \\
&\quad + Ferm(LH) + \textbf{ Ferm(CH) + b2(HH) + Ouv(CH)} \\
&\quad + \textbf{ Ferm(HH)} + b3(CH) + Ferm(CH) \\
CO_{S2} &= Ouv(LH) + Ouv(CH) \\
&\quad + b1(LH) + b1(CH) \\
&\quad + Ferm(LH) + \textbf{ b2(CH)} \\
&\quad + b3(CH) + Ferm(CH)
\end{aligned}
$$

avec :

- $Ouv(X)$: la quantité de produit (inutilement) pulvérisé lors de l'ouverture de la buse X. Cette quantité est en fait indépendante de la buse considérée. Nous laissons cependant l'information concernant la buse pour une meilleure compréhension de l'équation.
- $Ferm(X)$: idem pour la fermeture de la buse X.
- $bi(X)$: quantité de produit pulvérisé pendant le bloc bi par la buse X.

Ainsi, en fonction de la durée des blocs et de la succession des commandes, il est possible que l'utilisation d'une commande alternative soit plus économique pour certains blocs que l'emploi de la commande C_{best}. Pour notre exemple, on a $b2(CH) < Ouv(HH) + Ferm(CH) + b2(HH) + Ouv(CH) + Ferm(HH)$ si l'on suppose que toutes les mains ont le même débit. Dans ce cas, on peut donc en conclure que la commande C_{alt} est plus intéressante à utiliser que C_{best} pour le bloc b_2.

Pour résumer l'intérêt de la commande alternative, il est important de souligner que dans cette application du model-checking avec optimisation, la pulvérisation d'une quantité suffisante de produit à tout instant et sur chaque plant de vigne est synonyme de sécurité pour la production. Du fait de cette contrainte de sécurité qui engendre localement une pulvérisation plus importante lorsqu'on ferme un organe de pulvérisation pour ouvrir un autre, la commande alternative permet d'optimiser la quantité pulvérisée sur l'ensemble de la parcelle tout en utilisant une commande adaptée sur chaque bloc.

A la fin de l'étape 1, on obtient donc une carte statique (fonction de correspondance PSM) des configurations de pulvérisation possibles en fonction des états de végétation possibles. Dans l'étape 2, une carte de préconisation va être établie, c'est à dire un lien entre la végétation à chaque endroit du champ et les commandes C_{best} C_{alt} appropriées.

4.3.2 Étape 2 : Algorithme pour l'identification et la caractérisation des blocs de végétation

L'étape 2 consiste à traiter les données LiDAR réelles pour obtenir une liste de blocs de végétation pour le rang étudié, en utilisant la fonction de correspondance PSM détaillée à l'étape 1. Chaque bloc est caractérisé par ses commandes C_{best} et C_{alt}, et par la durée nécessaire à la pulvérisation dans ce bloc. Dans la suite de ce chapitre, par abus de langage et pour plus de concision, nous parlerons de durée d'un bloc. Le schéma-bloc de l'étape 2 est décrit par la figure 4.6.

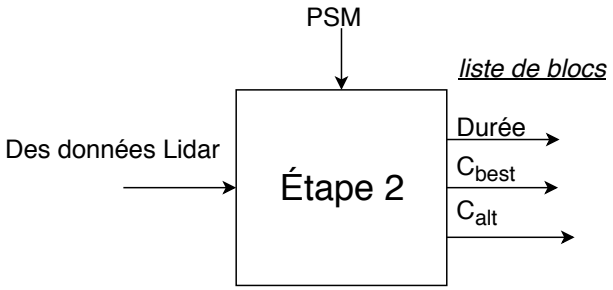


FIGURE 4.6 – Étape 2 : Calcul de C_{best}, C_{alt} et durée des blocs

La durée des blocs est une notion importante. Le bloc doit être suffisamment long pour garantir l'ouverture et la fermeture des buses au bon moment, mais des durées trop longues ne permettront pas de discriminer efficacement la densité de végétation. Dans notre cas, nous avons tout d'abord analysées les données LiDAR par tranches de largeur réduite (10 cm), ce qui permet de détecter avec une bonne précision l'absence ou la présence de végétation.

Par contre, il n'est pas possible dans notre cas d'application concret de considérer des blocs de végétation de seulement 10 cm. En effet, à une vitesse de 1,4 m.s−1 (environ 5 km.h−1), 10 cm ne représentent qu'une durée de 0,07s. Sachant que la durée d'ouverture ou de fermeture des buses nécessitent 0,2s, il n'est pas possible d'atteindre la configuration nominale de pulvérisation. En revanche si la taille des blocs est d'au moins 50 cm alors nous pouvons garantir que la configuration de pulvérisation souhaitée sera atteinte. Il est donc nécessaire d'assembler les tranches de 10 cm en blocs d'au moins 5 tranches.

Nous avons donc établi un algorithme d'identification basé sur les règles de filtrage et d'agrégation suivantes :

1. **Découpage des données LiDAR :**

   - Tout d'abord, les données LiDAR issues du rang de végétation sont échantillonnées par tranches de 10 cm. Puis pour chacune d'elles, et pour

chaque section d'intérêt (T, H, M et L), le décompte du nombre d'interceptions de faisceaux LiDAR est effectué, puis converti en valeur qualitative sur 0, 1, 2, ce qui permet d'obtenir un état de végétation codé sous la forme T-H-M-L pour chaque tranche, en respectant les seuils choisis.

- Puis, afin d'avoir des tranches voisines plus semblables, la règle RAF1 (Règle d'Agrégation et de Filtrage n°1) est appliquée pour chaque tranche et chaque section horizontale dans la tranche :

  Règle RAF1 : Soit 3 tranches successives $i-1$, i et $i+1$, et soit une section horizontale donnée X $\in$ {haut, milieu, bas, total}, si l'état de végétation de la tranche de droite et de gauche sont les mêmes mais que celle du milieu est différent, alors on passe l'état de végétation de la tranche du milieu à cette valeur :
  Si $étatVeg\_X(i-1) = étatVeg\_X(i+1) \neq étatVeg\_X(i)$ alors
  $étatVeg\_X(i) := étatVeg\_X(i+1)$.

2. **Identification et agrégation des blocs "Trous" :** L'objectif est ensuite de séparer les blocs de végétation des blocs trous, pour lesquels la configuration de pulvérisation doit fermer toutes les buses. Cela passe entre autres par l'agrégation des tranches en fonction des valeurs de la commande C_{best} :

   - Pour chaque tranche de 10 cm, C_{best} est définie en utilisant la fonction PSM. Cette étape s'intéresse également à la définition des "trou"[3], qui seront associés à la commande "toutes mains éteintes".
   - L'équivalent de la règle d'agrégation 1 est alors appliqué, mais en considérant les valeurs de la commande C_{best}.
   - Puis les tranches contiguës ayant la même commande sont regroupées. Par conséquent, les tranches de végétation manquante adjacentes sont également regroupées en un seul bloc trou. Cela permet d'obtenir des trous de taille plus importante.
   - Il s'agit maintenant de traiter le cas où des tranches trop petites existent encore. L'application des règles d'agrégation RAF2 et RAF3 permet de grossir les blocs de type "trou" suffisamment pour pouvoir garantir le contrôle des buses :

     Règle RAF2 : Elle consiste à associer les tranches de taille 10 cm (résolution=10cm) au bloc voisin de plus grande taille. Cette règle ne peut pas remettre en cause la qualité de la pulvérisation. En effet, les tranches sont de taille très fines et associer une tranche au bloc voisin ne perturbera pas significativement la qualité de pulvérisation.
     Si $taille\_bloc[i] = résolution$ et $taille\_bloc[i-1] < taille\_bloc[i+1]$ alors (*fusion de i et i+1, et suppression du bloc i*).

3. On rappelle que, selon PSM, une tranche sera considérée comme un "Trou" si M=0.

Règle RAF3 : S'il existe un bloc entre 2 trous proches [4] et que la commande pour ce bloc utilise une seule main (bloc de faible densité) alors on considère ce bloc comme un trou. La règle d'agrégation 3 ne remet pas en cause significativement la qualité de la pulvérisation. En effet, cette situation spécifique peut souvent représenter un bruit dans les données LiDAR. Par exemple, le LiDAR a pu détecter autre chose que de la vigne. Pour cette raison, un tel bloc sera considéré comme un trou.
Si $C_{best}[i-1] = 0$, $C_{best}[i+1] = 0$ et $C_{best}[i] \in \{LH, HH, CH\}$ alors alors (*fusion de i-1, i et i+1 en un seul bloc trou, et suppression du bloc i*).

- A ce stade, nous avons détecté avec précision l'emplacement des trous. Parmi ces derniers, nous allons retenir les blocs de taille supérieures à 50cm. Les blocs de trous de taille inférieure à 50 cm ne sont pas considérés car ils ne permettent pas un contrôle efficace, dû au temps d'ouverture et de fermeture des buses.

3. **Agrégation des autres blocs :** Les étapes suivantes permettent de former des blocs de végétation entre deux blocs trous retenus (i.e. de taille supérieure à 50 cm). A ce stade, les opérations précédentes de découpage des données LiDAR sont répétées :

   - la végétation entre les blocs "trous" retenus est divisée en blocs de longueur minimale de 50 cm. Un nouveau comptage des interceptions LiDAR est effectué pour ces blocs et le seuillage qualitatif est appliqué pour obtenir les valeurs T-H-M-L.
   - pour chaque bloc, C_{best} et C_{alt} sont calculés en utilisant la fonction PSM.
   - les blocs qui ont la même commande C_{best} et C_{alt} sont regroupés.

   Enfin, en considérant une vitesse fixe du pulvérisateur tout au long du rang, les longueurs de blocs sont converties en durées de pulvérisation.

Notre algorithme a ainsi permis d'extraire une liste de blocs de végétation à partir des données LiDAR brutes et de la fonction PSM. Les durées et les commandes C_{best} et C_{alt} de chaque bloc seront utilisées dans le modèle présenté à l'étape 3 de notre méthodologie AMPS.

4.3.3 Étape 3 : Modélisation du problème d'optimisation de la pulvérisation

Le modèle PTA_AMPS développé pour optimiser la pulvérisation est basé sur un réseau de 7 automates temporisés de coût :

- L'automate de démarrage gère le début de pulvérisation dans le rang.

4. La distance max entre deux trous considérés comme proche peut être définie empiriquement.

- Les automates de buses représentent le comportement de chaque buse. Il faut donc 3 automates.
- L'automate d'anticipation assure la sélection de la commande pour le bloc suivant.
- L'automate de déplacement représente le déplacement du pulvérisateur à l'intérieur du rang de bloc en bloc.
- L'automate de contrôle des buses gère l'ouverture et la fermeture de chaque buse selon la commande de configuration de pulvérisation choisie.

Dans ce qui suit, chaque automate est détaillé et la synchronisation entre les automates est décrite par un schéma fonctionnel.

4.3.3.1 Description détaillée des modèles d'automate temporisé de coût

A) Description des constantes

On rappelle que les deux commandes possibles pour chaque bloc de végétation sont C_{best} et C_{alt}, résultant de l'application de l'étape 2 de la méthode AMPS aux données LiDAR.

tempo_bloc[] : est un tableau d'entiers qui permet de stocker la durée de chaque bloc.

$Commande1$[] : est un tableau d'entiers qui permet de stocker la commande C_{best} de chaque bloc. On associe à chacune des 7 commandes possibles un code. Le code 0 est associé à la commande $--$ (toutes les mains sont fermées), le code 1 est associé à la commande LH, le code 2 est associé à la commande CH et le code 4 est associé à la commande HH. Les codes sont additifs en fonction des mains activées. Par exemple, si la commande C_{best} du premier bloc est $LH\&HH$ alors le code est 5 et $Commande1[0] = 5$.

$Commande2$[] : est un tableau d'entiers qui permet de stocker la commande C_{alt} de chaque bloc.

Nb_bloc est une constante de type entier, elle représente le nombre de blocs formant le rang d'étude.

B) Description des variables

La variable *currentStateNozzle*[] est un tableau qui représente l'état courant des buses. Le codage de cet état sous la forme d'un tableau d'entiers est nécessaire pour pourvoir l'exploiter dans la fonction $Control()$. $currentStateNozzle[0]$ représente l'état de la buse LH, $currentStateNozzle[1]$ représente l'état de la buse CH, et $currentStateNozzle[2]$ représente celui de la buse HH. Si l'état est OFF alors $currentStateNozzle[i]$ vaut 0. Si l'état est *TransitionOfOpen* (ouverture) ou *TransitionOfClose* (fermeture) alors $currentStateNozzle[i]$ vaut 2. Si l'état est ON alors $currentStateNozzle$ vaut 1.

La variable *nextStateNozzle*[] est un tableau qui représente l'état des buses pour le prochain bloc. Deux cas sont possibles pour chaque buse, ou bien la buse sera ouverte (code 1) ou bien la buse sera fermée (code 0). *nextStateNozzle*[0] représente l'état suivant de la buse LH, *nextStateNozzle*[1] l'état suivant de la buse CH et *nextStateNozzle*[2] celui de la buse HH.

Choice[] : est un tableau d'entiers qui permet de stocker pour chaque bloc la commande choisie parmi C_{best} ou C_{alt} par *l'automate d'anticipation*. On associe à chaque commande possible un code. Pour la commande C_{best}, le code est 1 et pour la commande C_{alt}, le code est 2.

currentBlock est une variable de type entier qui représente le numéro du bloc de végétation courant.

nextblock est une variable de type entier qui représente le numéro du prochain bloc de végétation.

anticipe est une variable de type booléen qui permet d'éviter que *l'automate d'anticipation* boucle infiniment.

allClose est une variable de type booléen, elle vaut `vrai` lorsque toutes les buses sont fermées et `faux` sinon.

C) Description des canaux de synchronisation

Les canaux multiples[5] *:*

endRow : ce canal permet à l'*automate de déplacement* d'envoyer un signal pour indiquer la fin du rang.

$finishAnticipation$: ce canal correspond au signal envoyé par l'*automate d'anticipation* pour indiquer qu'une commande a été sélectionnée pour le premier ou le suivant bloc.

Les canaux binaires :

*start*0 : ce canal correspond au signal envoyé par *l'automate de démarrage* à l'*automate d'anticipation* pour que celui-ci sélectionne une commande pour le premier bloc.

*start*1 : ce canal correspond au signal envoyé de *l'automate de démarrage* à l'*automate de contrôle des buses* pour activer les buses pour le premier bloc.

new_Block : ce canal correspond au signal envoyé par l'*automate de déplacement* à *l'automate de contrôle des buses* à la fin de chaque bloc de végétation.

new_block_dem : ce canal correspond au signal envoyé par *l'automate de démarrage* à l'*automate de déplacement* pour déclencher le déplacement pour le premier bloc.

endRow : ce canal correspond au signal envoyé par l'*automate de déplacement* à *l'automate de démarrage* pour indiquer la fin du rang.

$on/offNozzle$[] : ces canaux correspondent aux signaux envoyés par *l'automate*

5. Les canaux sont décrits chapitre 2, section 2.6.2.

de contrôle des buses aux automates de buses pour déclencher l'ouverture ou la fermeture des buses.

D) Description des fonctions

updateNextNozzle() : cette fonction prend en entrée le code de la commande et en fonction de la valeur de ce code, elle met à jour la variable *nextStateNozzle*. Par exemple, si la valeur du code est 5 alors la commande sélectionnée pour le bloc suivant est $HH\&LH$, et la fonction met à jour les variables aux valeurs suivantes : $nextStateNozzle[0] = 1$, $nextStateNozzle[1] = 0$ et $nextStateNozzle[2] = 1$.

control() : cette fonction détermine, à partir de la commande sélectionnée, quelles buses doivent être ouvertes et celles qui doivent être fermées pour le bloc suivant. Par exemple, si on a la situation suivante : $currentStateNozzle[0] = 0$, $currentStateNozzle[1] = 1$, $currentStateNozzle[2] = 1$, $nextStateNozzle[0] = 0$, $nextStateNozzle[1] = 0$ et $nextStateNozzle[2] = 1$ alors pour le prochain bloc, il faut fermer la buse du centre.

E) Description des automates :

L'automate de démarrage : cet automate, représenté sur la figure 4.7, gère la phase de démarrage. Il envoie les trois événements de commande *"start0"*, *"start1"* et *"new_block_dem"*.Il reçoit deux évènements *"finishAnticipation"* et *"endRow"*.

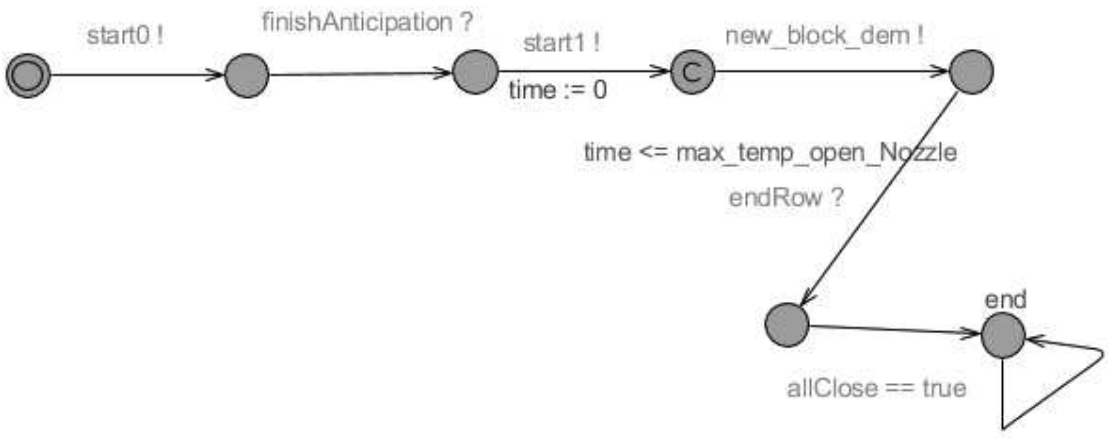


FIGURE 4.7 – L'automate de démarrage

Automate d'anticipation : cet automate, représenté sur la figure 4.8, permet de sélectionner une commande pour le bloc suivant. Les deux commandes possibles pour chaque bloc de végétation sont C_{best} et C_{alt} qui sont stockées respectivement dans $Commande1$ et $Commande2$. *L'automate d'anticipation* reçoit d'abord le signal *"start0"* de *l'automate de démarrage*. Ensuite, il choisit une commande parmi $Commande1$ et $Commande2$ si la valeur de la variable *Choice* est -1 (choix non

prédéfini) et stocke son choix dans cette variable *Choice* puis appelle la fonction *updateNextNozzle*. Si *Choice* est prédéfinie, alors ce choix est suivi. Enfin, *l'automate d'anticipation* envoie le signal *finishAnticipation* et remet à `false` la variable *anticipe* pour ne pas boucler indéfiniment.

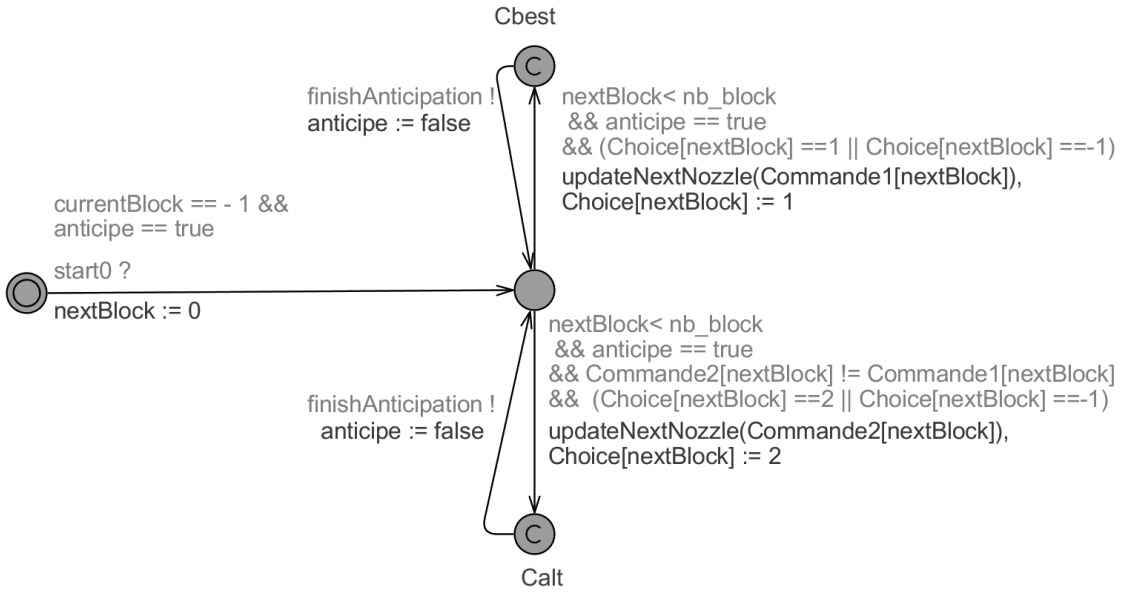


FIGURE 4.8 – L'automate d'anticipation

Automate de déplacement : cet automate, représenté sur la figure 4.9, permet de représenter le déplacement réel du pulvérisateur. Le temps nécessaire pour la pulvérisation d'un bloc de végétation est calculé au cours de l'étape 2, à partir de la longueur du bloc et de la vitesse du pulvérisateur. Cette information est stockée dans la variable *tempo_block*. Au début, *l'automate de déplacement* est dans l'état *"Init"*. Lorsqu'il reçoit le signal de synchronisation de démarrage *"new_block_dem"*, il passe à l'état *block*. Il restera dans cet état toute la durée du traitement de ce bloc : l'état *block* a un invariant $t\_block <= tempo\_block[currentBlock]$ et une garde $t\_block == tempo\_block[currentBlock]$ sur ses transitions sortantes. L'automate signale la fin d'un bloc par le signal *new_block* et la fin d'un rang par le signal *endRow* (il repasse alors dans l'état $Init$).

Automate de contrôle des buses : le rôle de cet automate, décrit sur la figure 4.10, est de contrôler les buses. Lorsqu'il reçoit le signal *"finishAnticipation"*, il appelle la fonction *control()* qui détermine les buses devant être ouvertes ou fermées pour le bloc suivant. Pour sécuriser la quantité de produit pulvérisé dans chaque bloc, les buses sélectionnées pour l'ouverture sont ouvertes 0,2s avant le début du bloc (la valeur 0.2 est stockée dans la constante $max\_temp\_open\_Nozzle$). Inversement, celles qui doivent être fermées ne le seront complètement que 0,2s après le change-

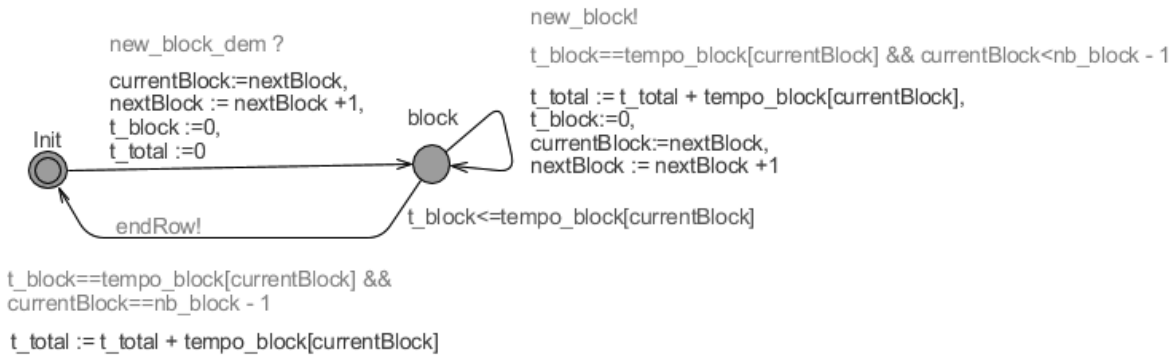


FIGURE 4.9 – L'automate de déplacement

ment de bloc (la valeur 0.2 est stockée dans la constante $max\_temp\_close\_Nozzle$). Pour ouvrir la buse i, l'*automate de contrôle des buses* envoie l'événement *onNozzle[i]* $max\_temp\_open\_Nozzle$ avant la fin du bloc courant (garde $t\_block <= tempo\_block[currentBlock] - max\_temp\_open\_Nozzle$), et pour la fermer il envoie l'événement *offNozzle[i]* après avoir reçu l'événement de synchronisation *new_block* de *l'automate de déplacement*. Dès que les commandes sur les buses sont effectives, alors la variable *anticipe* est mise à $True$ pour permettre l'anticipation pour le prochain bloc.

Automates de buses : ces automates représentent le comportement de chaque buse (figure 4.11). Chacun de ces automates possède quatre états : *OFF* (état initial), *ON*, *TransitionOfOpen* (ouverture) et *TransitionOfClose* (fermeture). Il reçoit les signaux *onNozzle* et *offNozzle* pour passer d'un état à un autre. Les temps de réponse des buses, pour l'ouverture ou la fermeture, sont supposés fixes et connus ($max\_temp\_open\_Nozzle$, $max\_temp\_close\_Nozzle$). Ils sont fixés à 0.2s dans notre exemple.

L'automate compte aussi la quantité de produit pulvérisé en utilisant la fonction de taux d'évolution de coût $cost'$ dans les états *TransitionOfOpen*, *ON* et *TransitionOfClose*. Cette fonction, spécifique de UppAal-CORA, représente dans ce modèle le flux de produit pulvérisé à tout moment par la buse considérée. Plus ce flux (débit) est élevé, plus le coût final, qui représente la quantité totale de produit pulvérisée, sera élevé. On peut noter que pendant les états *TransitionOfOpen* et *TransitionOfClose*, le flux n'est pas nul. En effet, on représente les transitions, pendant lesquelles les buses ne sont pas fermées, par des débits réduits de moitié. Puisque la variable $cost'$ doit être entière, $cost'$ vaut 2 à l'état *ON* et 1 dans les états de transition. UppAal-CORA. additionne les cost' de tous les automates du réseau d'automates. Dans notre cas, la quantité totale de produit pulvérisé correspondra à la somme des cost' des 3 automates buses.

4.3.3.2 Description de la synchronisation entre les automates

La synchronisation entre les automates est décrite par les deux diagrammes de séquence représentés figures 4.12 et 4.13, obtenus en utilisant le simulateur d'UppAal. Le premier diagramme gère le démarrage de la pulvérisation et le deuxième représente un exemple du fonctionnement en pleine végétation.

Au début *l'automate Demarrage*, qui est une instance de *l'automate de démarrage*, envoie le signal *start0* à l' *automate Anticipation*. A la réception de ce signal, ce dernier sélectionne la commande à activer pour le premier bloc et envoie le signal *finishAnticipation*. A la réception de ce signal, l'automate *Demarrage* envoie un signal *start1* à l'automate *controleurBuses* pour gérer l'ouverture et la fermeture des buses. Si, comme ici, le premier bloc est un trou, aucune buse n'est activée. Enfin, l'automate *Demarrage* envoie un signal *new_block_dem* pour indiquer le début du premier bloc.

Sur cet exemple en pleine végétation, les buses *HH* et *LH* sont déjà activées (Les automates *BuseBasse* et *BuseHaute* sont à l'état *ON*). L'automate *Anticipation* sélectionne la commande C_{best}, ici *(CH & LH)*, et envoie le signal *finishAnticipation* à l'automate *controleurBuses*. 0.2s avant le début du nouveau bloc, l'automate *controleurBuses* le signal *onNozzle* à l'*automate BuseCentre* pour l'ouverture de la buse. Cet automate passe de l'état *OFF* à l'état *TransitionOfOpen*. Lorsque *l'automate de déplacement* envoie le signal *new_block*, *l'automate controleurBuses* envoie un signal *offNozzle* à l'automate *BuseHaute* pour fermeture de la buse. Cet automate passe de l'état *On* à l'état *TransitionOfClose*.

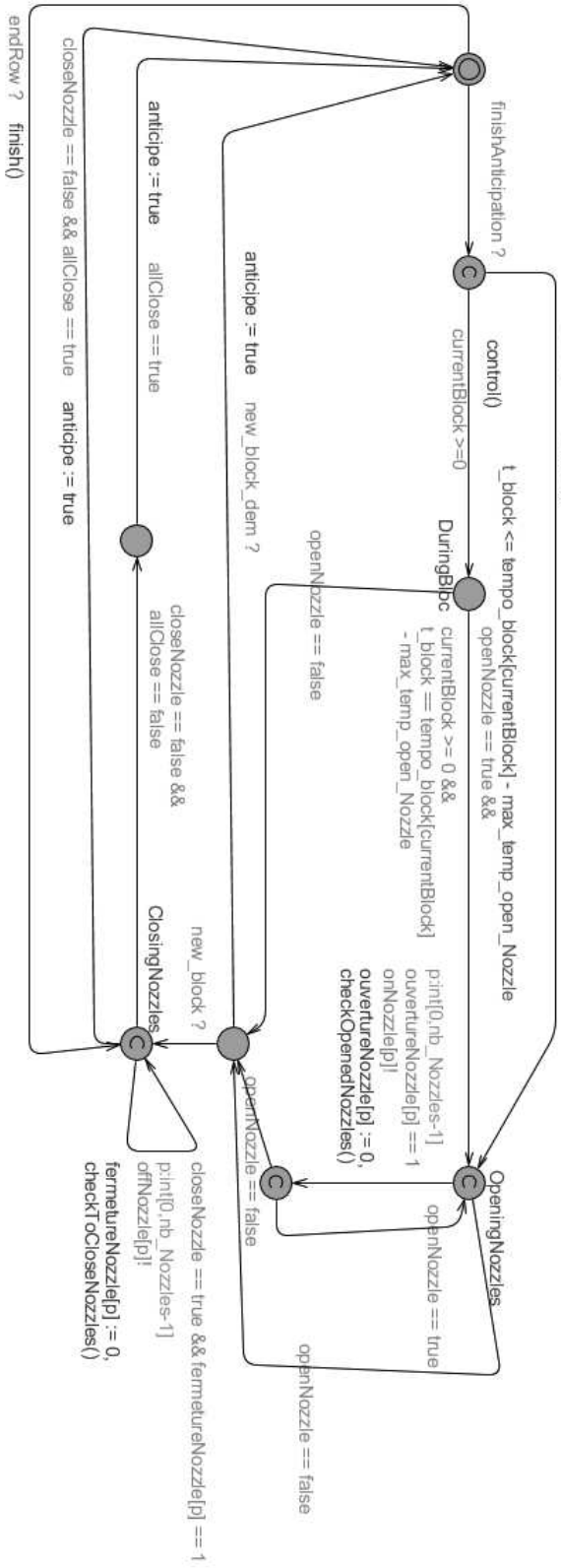


FIGURE 4.10 – L'automate de contrôle des buses

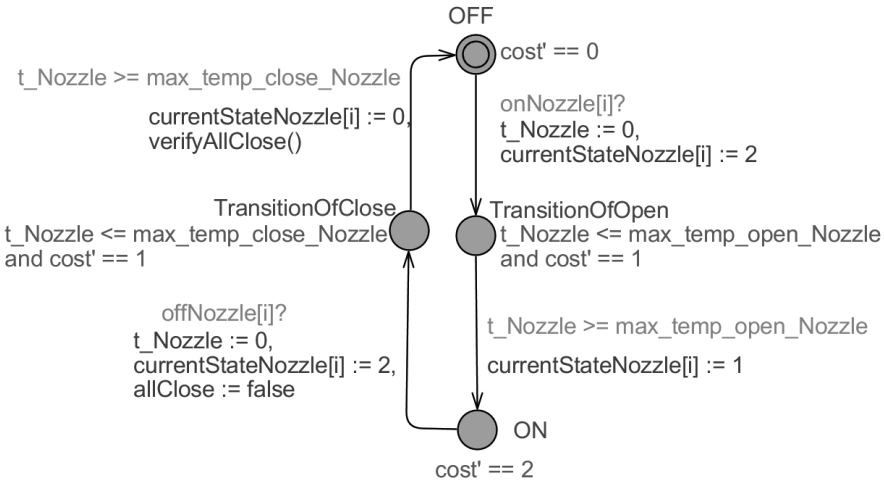


FIGURE 4.11 – L'automate de buse

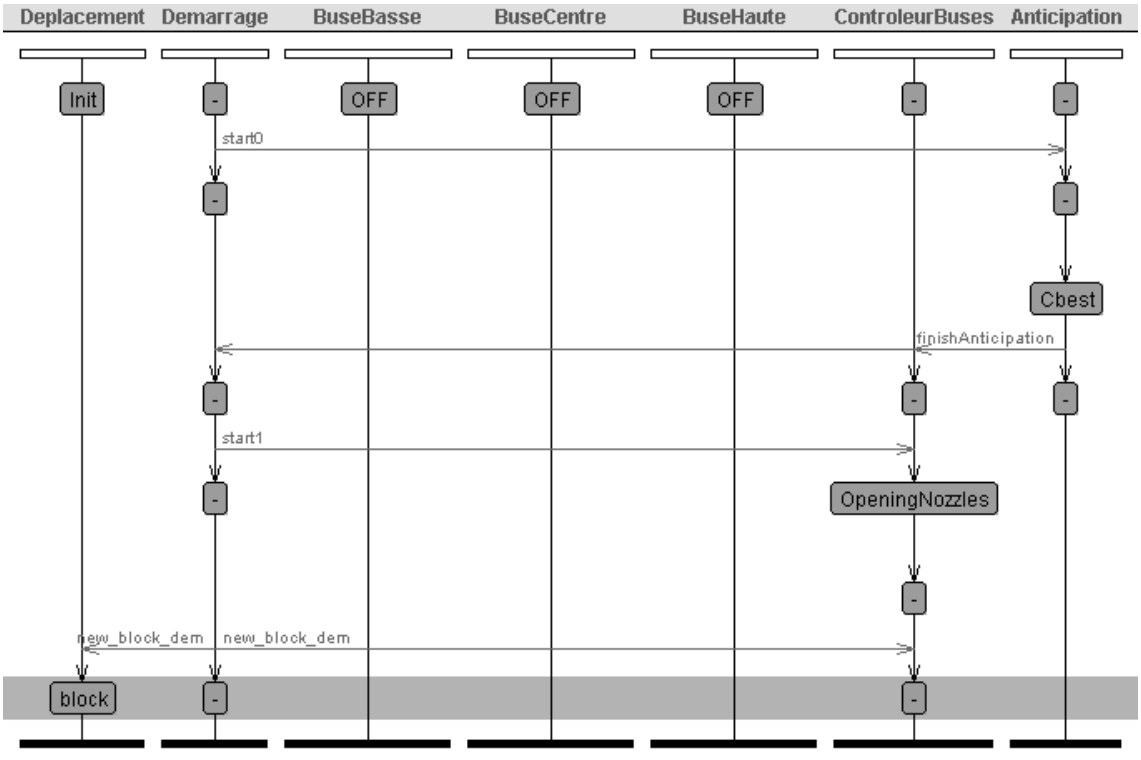


FIGURE 4.12 – Diagramme de séquence de démarrage de pulvérisation

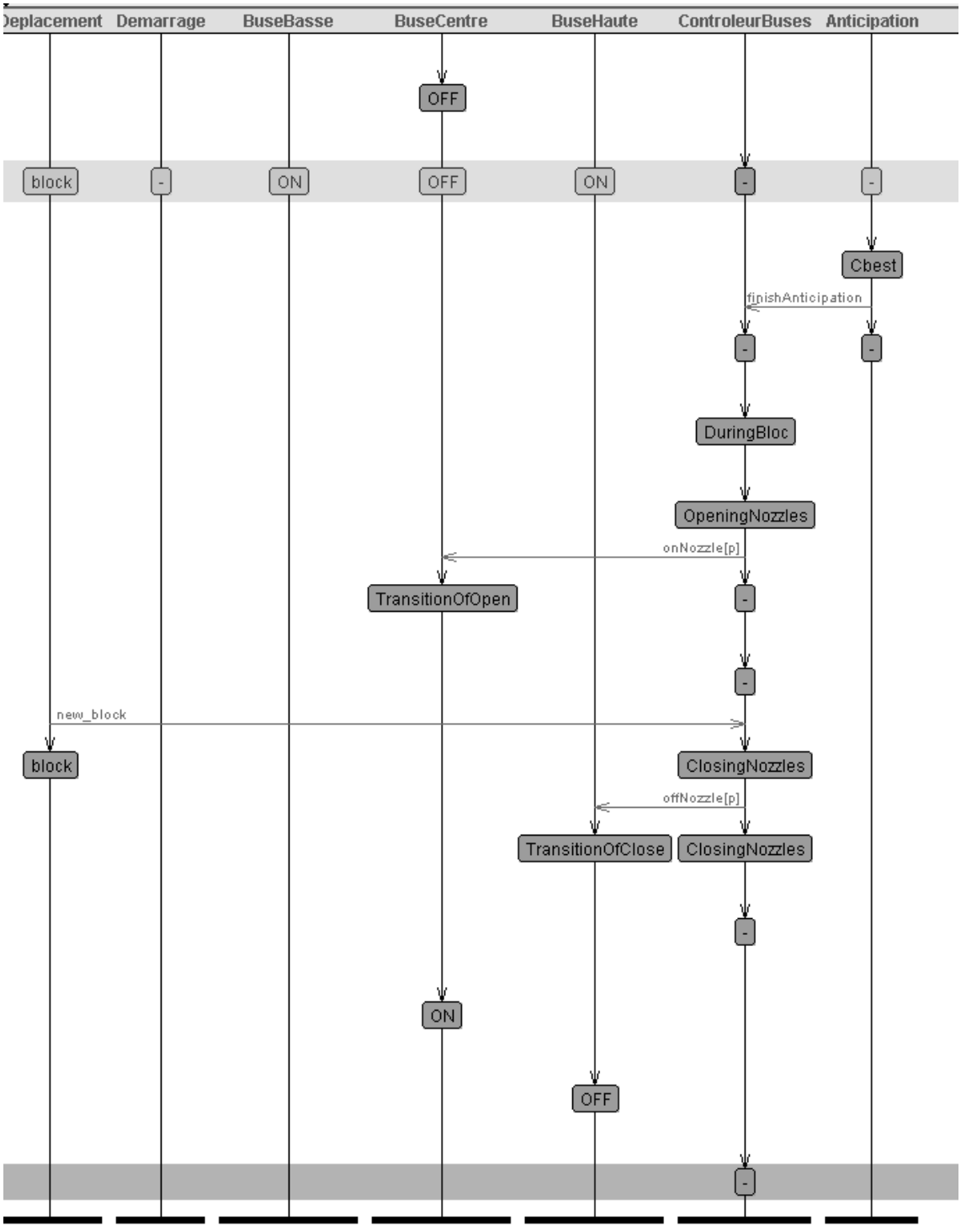


FIGURE 4.13 – Un exemple du fonctionnement en pleine végétation

4.3.4 Étape 3 (suite) : Vérification du modèle et calcul de la séquence de contrôle à l'aide d'UppAal-CORA

Comme déjà mentionné dans le chapitre 2, la technique du model-checking permet, dans un premier temps, de vérifier que le réseau d'automates a été bien conçu. Dans un second temps, nous pourrons calculer et vérifier la séquence de contrôle qui permet de pulvériser tout le rang tout en respectant l'ensemble des contraintes du système. La vérification s'effectue en vérifiant des propriétés, exprimées en logique temporelle, sur un modèle.

De nombreuses propriétés peuvent être vérifiées sur ce modèle pour s'assurer de sa cohérence et de son comportement. Dans la suite nous proposons quelques exemples et nous les formalisons en logique temporelle CTL.

1. Est-il possible de traiter tout le rang en utilisant toujours la commande définie comme la plus adaptée localement (choix systématique de C_{best}) ? Réponse attendue : Oui. Cette propriété peut se formaliser de la façon suivante :

PP 1 : $E <> \ forall(i \ : \ int[0, nb\_block-1])\, Choice[i] \ == \ 1 \, and \, ProcessVeg.end$

2. Est-il possible de traiter tout le rang en utilisant toujours la commande alternative "acceptable" C_{alt} ? Réponse attendue : Oui. Il est possible de la formuler comme suit :

PP2 : $E <> \ forall(i \ : \ int[0, nb\_block-1])\, Choice[i] \ == \ 2 \, and \, ProcessVeg.end$

3. Y a-t-il une séquence de contrôle qui permet au pulvérisateur d'atteindre la fin du rang (tout en appliquant les contraintes de contrôle spécifiées dans le modèle) ? Cette propriété permet d'atteindre la fin du rang sans imposer de choisir systématiquement C_{best} ou C_{alt}. La séquence de contrôle qui sera obtenue peut être un mélange de contrôle localement optimal (C_{best}) et de contrôle local alternatif (C_{alt}). Cette propriété se traduit en logique temporelle par :

PP3 : $E <> currentBlock == nb\_block - 1 \, and \, ProcessVeg.end$

La propriété qui va servir pour obtenir la séquence de contrôle optimale du point de vue du coût est la propriété 3. Nous appelons cette séquence de contrôle la *commande optimale*. UppAal-CORA vérifie cette propriété et fournit une trace avec le coût le plus bas. La fonction de coût calcule la quantité totale de produit pulvérisé sur le rang. Ainsi, le coût optimal calculé est la quantité minimale de produit nécessaire pour pulvériser le rang tout en assurant une protection suffisante.

4.4 Résultats expérimentaux de la méthode AMPS

Dans cette section, nous évaluerons la faisabilité de l'application de la méthode AMPS avec des données LiDAR acquises sur un rang d'une parcelle réelle de vigne

tout en tenant compte de la dynamique du pulvérisateur. La séquence de contrôle obtenue grâce à notre méthode (i.e. la commande optimale) sera comparée à la pulvérisation classique sans automatisation.

4.4.1 Description du rang d'étude

Dans la mesure où le pulvérisateur coupe les buses pendant les manœuvres en fin de rang, les séquences de commande de chaque rang peuvent être calculées indépendemment. Pour évaluer la faisabilité de l'approche, nous l'avons donc appliquée sur les données d'un rang unique.

La figure 4.14 illustre le rang étudié (ligne jaune dans la parcelle). La vigne étudiée est une parcelle de cépage Chardonnay en pleine végétation. Elle est située dans le sud de la France, au domaine Mas Piquet. Le rang de vigne a les coordonnées suivantes :

- Début du rang : latitude = 43°39.5625050'N et longitude = 3°49.5774463'E
- Fin du rang : latitude = 43°39.5670393'N et longitude = 3°49.5660410'E

Ce rang a été scanné avec un LiDAR 2D tel que présenté dans [10].

FIGURE 4.14 – Le rang de vigne de cépage Chardonnay étudié

Le rang fait 180 mètres de long. Le profil de végétation du rang complet est présenté dans la figure 4.15. Ce profil est condensé. Un zoom sur les 20 premiers mètres du profil de végétation du rang étudié est présenté dans la figure 4.16. Le nombre d'interceptions LiDAR y est représenté en niveau de vert. Ce profil contient quelques blocs de type "Trous" (boîtes jaunes). Par exemple, le bloc de végétation encadré par x = [5,8m : 7,2m] et y = [0,6m : 1,1m] est un bloc de végétation manquante. Il y a aussi beaucoup de cas où la valeur 0 doit être assignée à la section H (les zones sans végétation au-dessus de 1.2m, montrées avec la couleur orange).

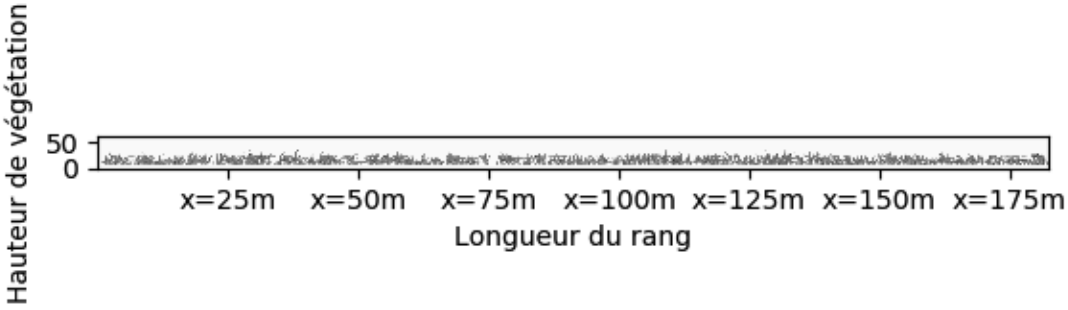


FIGURE 4.15 – Profil de végétation du rang étudié

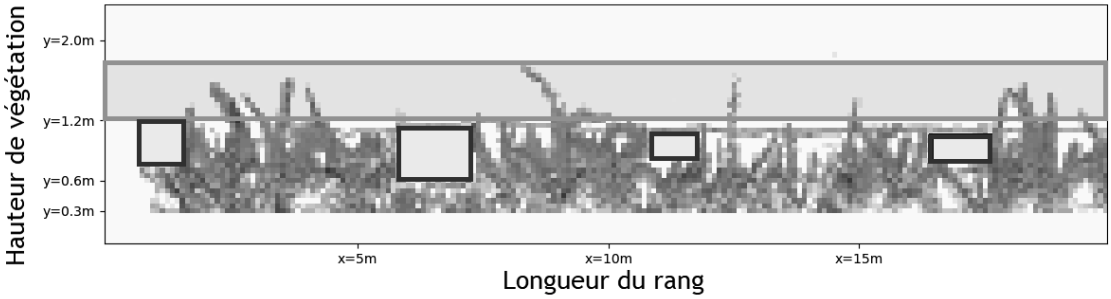


FIGURE 4.16 – Extrait du profil de végétation étudié

Dans la suite, la méthode AMPS sera mise en oeuvre sur le rang d'étude. L'étape 1 de AMPS consiste à définir une carte de correspondance entre les états de densité de la végétation et les configurations de pulvérisation souhaitées pour cet état. Nous avons déjà donné cette carte dans la section 4.3.1 (se référer au tableau 4.2). Nous passons alors à appliquer l'étape 2 de la méthode AMPS au rang d'étude.

4.4.2 Étape 2 : Détermination des blocs et des configurations de pulvérisation envisageables

Dans cette étape les données LiDAR du rang que l'on vient de décrire vont être traitées pour obtenir C_{best}, C_{alt} et la durée de chaque bloc de végétation. Au début, le rang est formé de 1817 blocs de végétation de taille 10 cm. En appliquant l'algorithme proposé, le nombre de blocs est réduit à 262 blocs dont 10 blocs de type "Trous". La longueur d'un bloc est exprimée ici en fonction du temps nécessaire pour le pulvériser. La plus petite durée d'un bloc est de 0,4s, ce qui correspond à une longueur de 50 cm et la plus longue est de 1,8s, ce qui correspond à une longueur de 2,5m. 72% des blocs ont la plus petite durée, et 8% des blocs ont une durée supérieure à 1s.

Un histogramme du nombre de blocs par durée est donné dans la figure 4.17. On remarque que le nombre de blocs dont la durée est 0,4s est 190 blocs et le nombre de blocs dont la durée est supérieure à 1s est 20 blocs. En plus de leur durée, les blocs

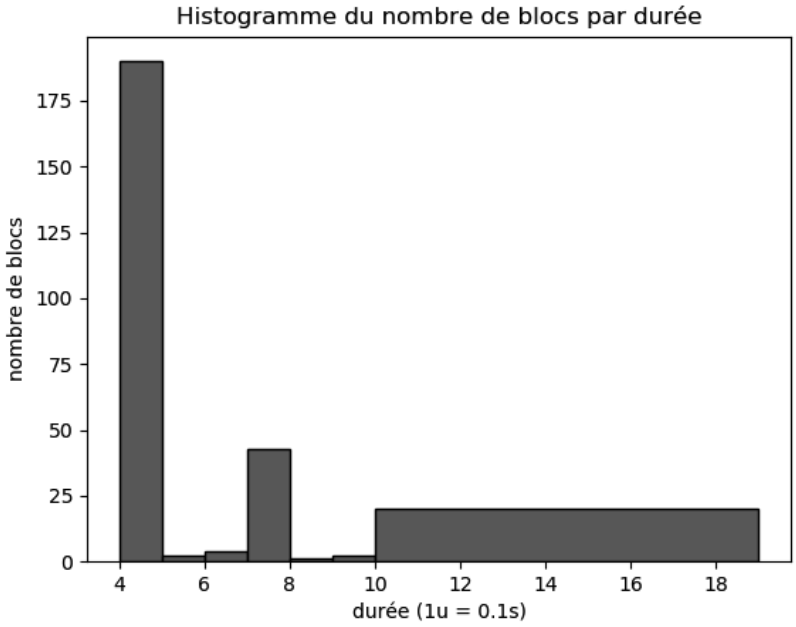


FIGURE 4.17 – Extrait du profil de végétation étudié

obtenus sont caractérisés par leurs commandes C_{best} et C_{alt}. La carte obtenue de ces commandes pour la liste des 262 blocs est donnée à la figure 4.18, en représentant les blocs spatialement et par leur durée.

A chaque type de commande, une couleur est affectée : LH (bleu), CH (rouge), HH (vert), LH & CH (magenta), LH & HH (cyan), CH & HH (jaune) et - (blanc) lorsque toutes les mains sont éteintes. La largeur de la commande est la durée du bloc. Pour les commandes C_{best}, les configurations les plus rencontrées sont LH & CH et LH & HH et pour les commandes C_{alt}, les configurations les plus rencontrées sont LH & HH, LH & CH et CH.

Il faut noter que cette figure ne représente pas l'exécution finalement choisie, avec l'état des buses. Elle représente les commandes C_{best} et C_{alt}. C'est le choix par optimisation qui permet de déterminer l'exécution choisie pour tout le rang.

Pour expliquer en détails le lien entre le profil de végétation et les commandes C_{best} et C_{alt}, nous considérons l'exemple suivant.

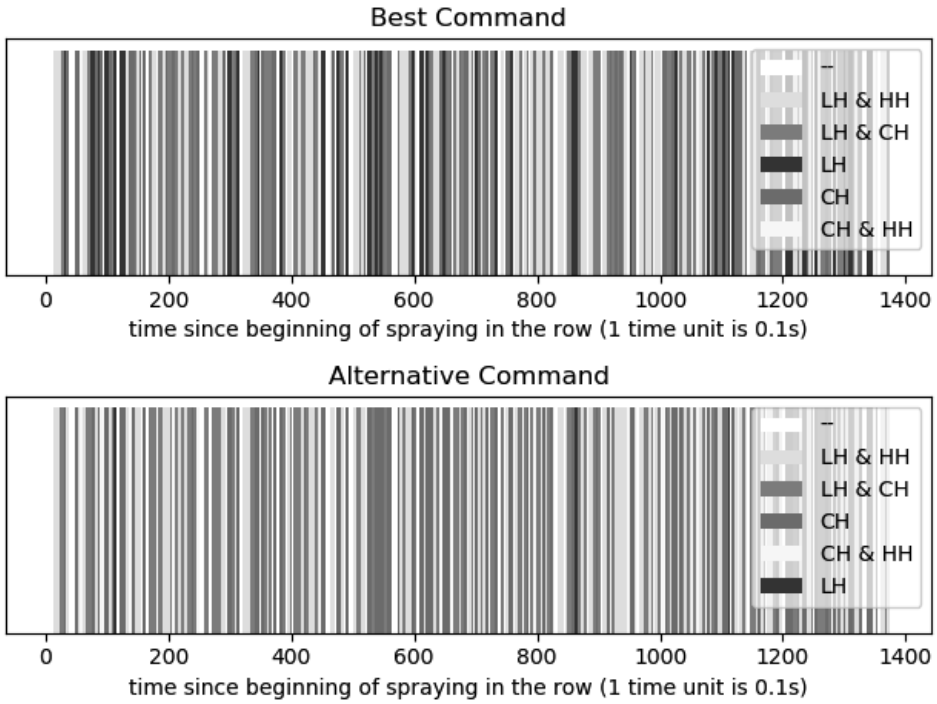


FIGURE 4.18 – Sortie de l'étape 2 : C_{best} et C_{alt} pour les blocs du rang étudié

Soit *Partie*1 une partie du rang étudié formée par les 10 premiers mètres de notre rang exemple. La figure 4.19 présente son profil de végétation.

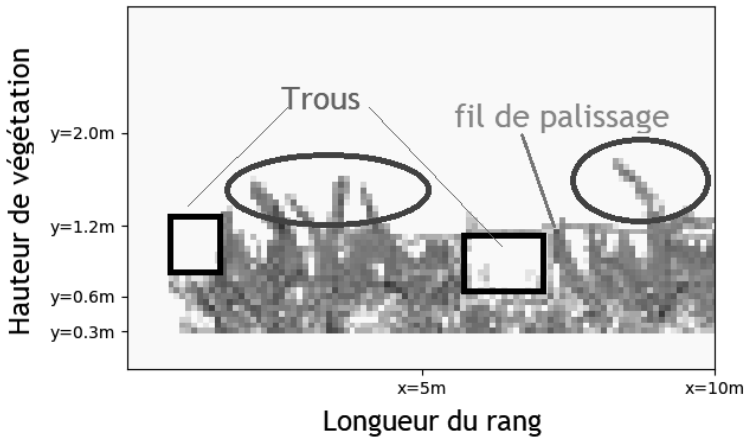


FIGURE 4.19 – Le profil de végétation pour la partie *Partie*1

On remarque que ce profil contient deux blocs de type trous (rectangle en rouge) : un au début et un au milieu. On rappelle que la végétation située en dessous de 0.3m

n'est pas considérée comme de la vigne et ne doit pas être traitée. Puis on trouve deux grands types de zones de végétation : l'un avec de la végétation au milieu et en bas, pour lequel potentiellement une seule main peut suffire pour la pulvérisation, et l'autre avec en plus un peu de végétation au dessus du fil de palissage (cercles en bleu), pour lesquelles il faudra a priori utiliser deux mains pour la pulvérisation. L'étape 2 appliquée sur la $Partie1$ permet de la décomposer en 14 blocs de végétation de densité homogène. La carte des commandes C_{best} et C_{alt} obtenue est donnée à la figure 4.20.

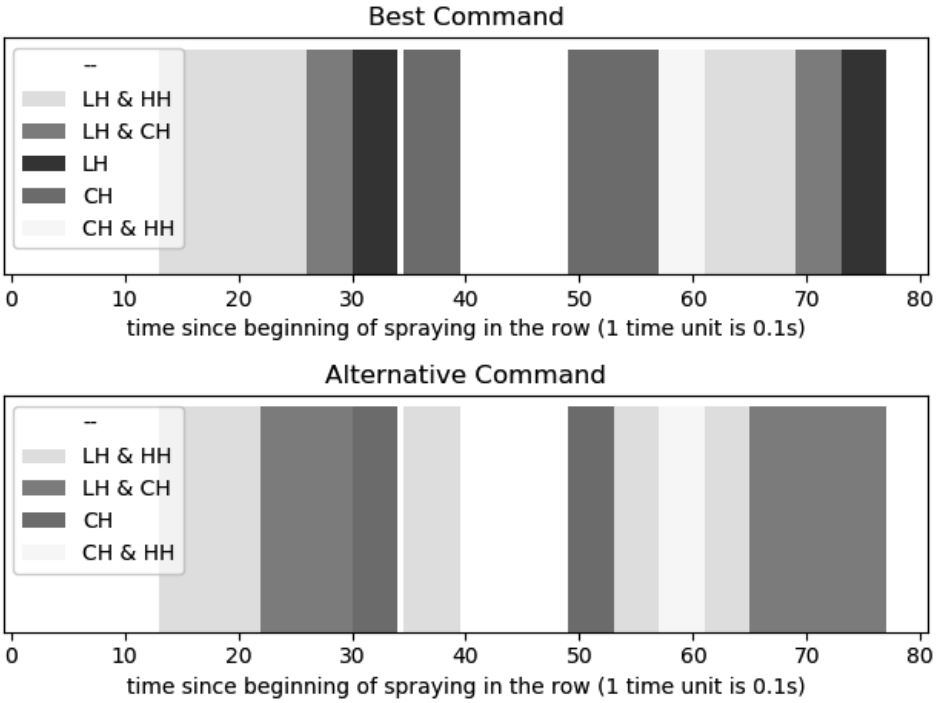


FIGURE 4.20 – Sortie de l'étape 2 : C_{best} et C_{alt} pour $Partie1$

D'après la figure 4.20, certaines zones ont des profils de végétation pour lesquelles les commandes C_{best} et C_{alt} sont identiques car incontournable : pour les deux trous la commande nulle $--$ (couleur blanche) ; pour les deux zones avec de la végétation au dessus du fil de palissage (entourées en bleu) et de la végétation dense en bas, alors il faut sélectionner une commande qui permet de couvrir toute la hauteur de végétation (LH & HH ou LH & CH) ; et pour la zone après le deuxième trou avec une végétation au milieu pour laquelle CH est sélectionnée. Dans les autres cas, il existe deux solutions de pulvérisations acceptables, en respectant la logique de pulvérisation (une ou deux mains, en haut ou en bas) en fonction de la hauteur et de la densité de végétation.

On remarque que le passage du profil de végétation aux commandes est cohérent.

Maintenant, à partir des données de sortie de l'étape 2, nous allons calculer la séquence de contrôle optimale choisissant pour chaque bloc soit C_{best} soit C_{alt} et minimisant globalement la quantité de produit de pulvérisation.

4.4.3 Étape 3 : Détermination de la séquence de contrôle optimale

Nous rappelons que la commande optimale est la séquence de contrôle correspondant aux contraintes du système et ayant le coût global le plus bas. Cette séquence est obtenue par la vérification de la propriété 3 [6] sur le modèle PTA_AMPS en utilisant l'outil UppAal-CORA pour optimiser le coût.

Lors de la vérification, nous n'avons pû trouver la commande optimale que pour 40 blocs. Au delà, un problème d'explosion combinatoire se produit après 2 à 3 minutes du début de la vérification. Deux raisons expliquent l'apparition de ce problème : la première est liée à la complexité intrinsèque du model-checking. En effet, en simplifiant, le rang de vigne étudié est formé de 262 blocs et pour chaque bloc deux commandes sont possibles ce qui fait une complexité de 2^{262}. La complexité de décider entre ces deux commandes pour chaque bloc d'un rang est exponentielle en ce qui concerne le nombre de blocs. En effet, au niveau de l'arbre de décision chaque nouveau bloc induit la création de 2 nouvelles branches et pour trouver le meilleur coût, il faut construire tout l'arbre et cela explose. La deuxième raison est liée à une limitation technique de l'outil de model-checking UppAal-CORA. En effet, le binaire disponible de UppAal-CORA a été construit uniquement pour une architecture 32 bits ce qui limite l'utilisation de la mémoire à 4 Go de RAM. Calculer la séquence optimale de commande pour un rang formé de 262 blocs en utilisant moins de 4 Go de mémoire n'a donc techniquement pas été possible.

Les résultats que l'on a pû obtenir sont présentés dans le tableau 4.3. La colonne "Limite" permet de donner l'indice du premier et du dernier bloc formant la partie du rang vérifiée Qi. La colonne nommée "Coût" donne le coût optimal CO_i en quantité de produit pulvérisé pour la partie Qi. La colonne nommée "TE" (Temps d'Exécution) donne le temps total requis pour la vérification de la propriété, et la colonne nommée "M" (Mémoire) donne la mémoire requise pour la vérification de la propriété. Le temps et la mémoire nécessaire à la vérification ont été mesurés avec l'outil `memtime` décrit section 2.7.4.

Pour estimer l'efficacité de notre méthode AMPS, il est intéressant de faire une comparaison sur le coût (c'est à dire la quantité de pesticide consommée) avec un pulvérisateur classique pour estimer ce que l'on gagne. Le tableau 4.4 présente une comparaison des coûts entre un pulvérisateur classique et le pulvérisateur ciblé optimisé grace à l'AMPS. La colonne nommée "Coût Classique" donne le coût CC_i pour la partie Qi avec un pulvérisateur classique. Ce coût est calculé en suppo-

6. PP3 : $E <> currentBlock == nb\_block - 1$ *and* $ProcessVeg.end$

TABLE 4.3 – Coût optimal des parties formant le rang d'étude

Limite	Coût (CO_i)	TE (s)	M (KB)
Q1 = b0-b31	460	7.65	172 960KB
Q2 = b0-b40	652	108.08	2 641 760KB
Q3 = b0-b41	Out of memory	150.39	4 167 248

sant une utilisation traditionnelle du pulvérisateur, c'est à dire en utilisant les deux mains "LH" et "HH" en même temps et en permanence. La colonne nommée "Gain" représente le gain obtenu grâce à l'AMPS.

TABLE 4.4 – Comparaison du coût de pesticide entre un pulvérisateur classique et le pulvérisateur ciblé

Limite	Coût Optimal(CO_i)	Coût Classique (CC_i)	Gain
Q1 = b0-b31	460	668	31%
Q2 = b0-b40	652	852	23%

D'après les résultats du tableau 4.4, notre méthode permet d'économiser jusqu'à 31 % du produit par rapport un pulvérisateur classique ce qui est très intéressant. Le gain n'est cependant pas constant, car il dépend du profil de végétation des parties étudiées. Un premier élément réside dans le nombre de trous, dans lequel la commande dynamique permet de couper la pulvérisation. Dans notre exemple, le nombre de trous pour les parties Q1 et Q2 est identique (= 3), ce qui ne différencie pas les gains. Pour cela il faut regarder le nombre de fois où la commande optimale n'utilise qu'une seule main plutôt que deux pour la pulvérisation classique. Une analyse de la trace montre que la séquence optimale pour la partie Q1 utilise 20 fois une seule main pour 31 blocs, ce qui représente 65% du nombre de blocs. Pour la partie Q2, la séquence optimale utilise 22 fois une seule main pour 40 blocs, ce qui représente 55% du nombre blocs. Ceci explique que le gain pour la partie Q1 est plus grand que celui pour Q2.

Ainsi, notre méthode montre bien qu'elle est intéressante par rapport à un pulvériseur classique. Cependant, il faudrait pallier à l'explosion combinatoire afin de pouvoir vérifier la propriété d'atteignabilité sur tout le rang de vigne. Ce problème sera discuté par la suite.

4.5 Conclusion

Dans ce chapitre, nous avons étudié comment les techniques de model-checking et les automates temporisés de coût peuvent contribuer à l'agriculture de précision,

pour le développement de méthodes de commande afin d'optimiser une pulvérisation de précision en fonction de la densité de la végétation.

Nous avons proposé la méthode AMPS (Automate Modeling for Precision Spraying) qui prend en entrée des données terrain issues d'un scan LiDAR 2D de la canopée d'une parcelle et qui, en considérant la dynamique du pulvérisateur (ouverture et fermeture des buses), calcule la séquence de commande optimale pour la pulvérisation de la parcelle ciblée. Nous avons montré que cette méthode permettait d'optimiser la quantité de produit phytosanitaire consommé par rapport à une pulvérisation classique, et nous l'avons appliqué à un cas concret avec des données réelles issues d'une parcelle de vigne.

Cependant, au moment du calcul de la séquence de commande optimale, nous avons été confronté au problème d'explosion combinatoire, ce qui nous a empêché d'obtenir la séquence de commande optimale pour la totalité du rang de vigne étudié. Ce problème est dû d'une part à la nature du problème traité qui nécessite une exploration exhaustive de l'espace d'états, et d'autre part, à l'outil de vérification choisi qui est construit pour une architecture de 32 bits limitant la mémoire RAM utilisable à seulement 4 Go. Il est donc nécessaire d'explorer d'autres solutions. Nous avons étudié la piste d'une décomposition spatiale du problème, qui sera discutée dans le chapitre 8 de ce manuscrit.

Pour l'instant, dans le chapitre suivant, nous allons continuer à explorer les possibilités offertes par le model-checking pour l'agriculture de précision en développant une méthode adaptée au problème de vendange sélective.

5 Problème de vendange sélective

Sommaire

5.1 Introduction

Comme indiqué au chapitre 1, l'agriculture de précision (AP) est un sujet de recherche depuis de nombreuses années. Son objectif est d'adapter les actions agricoles à la situation locale sur le terrain afin que la bonne action soit effectuée au bon endroit et au bon moment [95]. Nous avons vu dans le chapitre précédent une application de cet objectif pour résoudre un problème de pulvérisation de précision dans un rang de vigne. La pulvérisation dans un rang de vigne correspond à une représentation spatiale que nous avions qualifié de '1D'. Dans ce chapitre, la représentation spatiale change et une nouvelle application des techniques d'agriculture de précision sera traitée. Lorsque les techniques d'agriculture de précision conduisent à préconiser des actions différenciées par zones, leur mise en œuvre peut nécessiter de résoudre des problèmes de logistique sur des représentations spatiales plus larges. La récolte sélective dans les vignobles en est un exemple, que nous avons qualifié de "1D+". Au lieu de tout récolter dans une parcelle pour obtenir la même qualité de vin, on aimerait pouvoir exploiter l'information directe ou indirecte que l'on a sur la qualité du raisin, avant la récolte, pour définir des zones de qualités différentes et réaliser une récolte différenciée pour obtenir un breuvage plus ou moins noble

On peut supposer qu'une carte géolocalisée distinguant 2 qualités de raisin est disponible avant la récolte. Ces cartes sont définies en fonction du rendement des vignes et de la qualité des raisins [34, 33, 81]. Il existe des services commerciaux qui proposent d'établir de telles cartes [1]. Le problème est alors de trouver la meilleure logistique de récolte, sous condition que des critères de qualité du raisin, exprimés en tant que contraintes sur le système de récolte, soient satisfaits. Cette logistique permet d'optimiser la durée de récolte et par conséquence réduire à la fois la quantité de carburant consommée et le coût du chantier de récolte.

Nous étudions dans ce chapitre comment UppAal-CORA, que nous avons précédemment utilisé pour l'optimisation de la commande de pulvérisation d'un rang de vigne, peut être appliqué à la logistique d'une vendange sélective de raisins cultivés en rangs. Ce problème a été précédemment abordé par Nicolas Briot ([35, 36, 37, 38]) avec des techniques de programmation par contraintes. Nous pensons qu'une approche basée sur UppAal-CORA pourrait avoir l'avantage sur d'autres approches alternatives d'optimisation (comme par exemple les méthodes de programmation en nombres entiers ou la programmation par contraintes), lorsque la dynamique du système étudié peut être décrite directement dans le formalisme de modélisation ATC (automates temporisés de coût). L'objectif de cette contribution n'est pas de proposer une méthode plus performante en temps de résolution que les méthodes déjà développées par N. Briot. Il est plutôt de montrer qu'un outil flexible comme UppAal-CORA, basé sur l'exploitation par la vérification-optimisation de modèles ATC, permet de décrire facilement les contraintes logistiques et dynamiques d'une

1. Par ex : `http://www.icv.fr/conseil-viticulture-oenologie/oenoview`

récolte, et de trouver efficacement des solutions optimales. La méthode proposée repose sur les mêmes bases que la méthodologie du chapitre précédent : la modélisation des contraintes et des coûts (ici liés au temps de récolte) sous la forme d'un réseau d'ATC avec UppAal-CORA qui sera ensuite employé pour vérifier une requête d'atteignabilité à coût minimal. Le contenu de ce chapitre a fait l'objet d'une communication à WODES en 2018 [123].

Le chapitre est organisé comme suit. Des éléments de fonctionnement sur la machine de récolte sélective sont donnés dans la section suivante. Puis, le problème de planification du trajet de la machine à vendanger pour une récolte sélective, que nous appelons problème de récolte sélective, ou problème de récolte différentielle (Differential Harvesting Problem, DHP), est détaillé. Dans la section 5.4, un modèle exprimant le DHP sous la forme d'un réseau d'automates temporisés de coût est décrit. La propriété d'accessibilité utilisée pour résoudre le DHP avec le modèle est ensuite définie. Avant de conclure, dans la section 5.5, les résultats expérimentaux sont décrits et discutés. Une conclusion clôt le chapitre.

5.2 Fonctionnement d'une machine à vendanger sélective

Comme on dispose d'une carte de la parcelle avec deux qualités de raisin, la question qui se pose maintenant est de savoir quel type de machine est capable de récolter et trier les deux qualités de raisins. Pour le DHP, nous considérerons une machine du type de celle équipée de `Eno Control` de `New Holland` (figure 5.1). Cette machine a eu la Médaille d'or au Sitevi [2] en 2009.

Une machine à vendanger est composée principalement d'une tête de cueillette et de trémies. La tête de cueillette de la machine à vendanger est composée d'un ensemble bras batteurs et de convoyeurs qui déplacent les raisins tombés de la vigne vers un mécanisme sur le dessus de la machine qui dirige les raisins vers les trémies (voir figure 5.2). Pour une machine à vendanger sélective, une trémie sera affectée au raisin de qualité supérieure et une autre trémie au raisin de moindre qualité. La capacité des trémies est identique et limitée à une valeur maximale que nous appellerons $CmaxHopper$. Généralement, lorsque l'une des deux trémies est pleine alors la machine vide ses deux trémies dans des bennes situées autour de la parcelle (on parle de "vidange"). Ces bennes sont transportées vers la cave immédiatement après chaque vidange afin de préserver la qualité des raisins. Elles sont immédiatement remplacées par d'autres bennes en attente sur la parcelle. On pourra donc considérer que les bennes sont toujours présentes. Pour garantir la bonne qualité de la récolte, il est préférable de faire la vendange en dehors des heures chaudes pour limiter l'exposition des raisins au soleil.

2. Salon International des équipements et savoir-faire pour les productions vigne-vin, olive, fruits-légumes

FIGURE 5.1 – Vue d'ensemble de la machine pendant la récolte

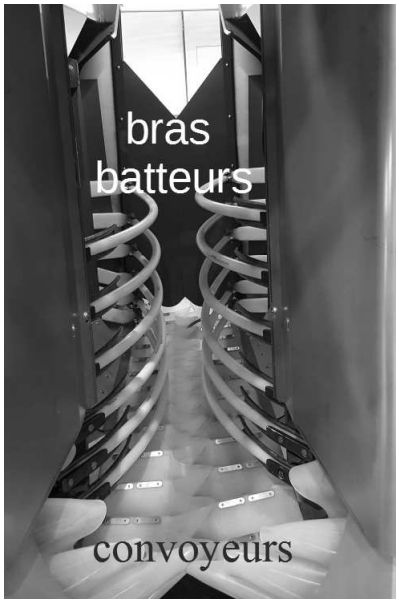


FIGURE 5.2 – Vue intérieure de la tête de cueillette

En raison de la taille longitudinale de la tête de cueillette, un phénomène appelé *latence* se produit lorsque la machine se déplace entre 2 zones de qualités différentes. Au moment de transition entre les zones et pendant un certain moment (la *latence*), les raisins sont mélangés et la qualité de raisin récoltée n'est pas définie. Le raisin ne

doit alors pas être dirigé vers la trémie réservée à la meilleure qualité. Le moment de *latence* dépend de la vitesse de la machine et de la taille de la tête de cueillette.

5.3 Définition du problème de vendange sélective

Dans cette section, nous introduisons le problème de vendange sélective, et plus spécifiquement le DHP, tel qu'il est défini dans [36]. L'intérêt de prendre cette définition est de pouvoir, par la suite, établir une comparaison qualitative de 2 approches traitant le même problème. Globalement, les variables de notre approche ont été définies pour être cohérentes avec [36].

5.3.1 Définition générale

Un champ de vigne est composé de plusieurs zones. Ces zones sont classées en fonction de la qualité de raisins qu'elles produisent. Deux qualités des raisins sont distinguées : la qualité A est la qualité supérieure et la qualité B désigne le reste des raisins. Chaque zone est donc étiquetée A ou B suivant la qualité des raisins qu'elle contient. Un même rang de vigne peut traverser plusieurs zones différentes (figure 5.10). Nous appelons dans la suite "raisins A" les raisins de qualité A, et "raisins B" les raisins de qualité B.

On dispose de la machine à vendanger décrite dans la section précédente. Nous appellerons cette machine HM (Harvesting Machine). HM permet la vendange sélective de deux qualités et dispose donc de deux trémies que nous appellerons $a-trémie$ et $b-trémie$. HM est capable de diriger les raisins vers la trémie de son choix et peut ainsi trier les qualités de raisins selon les zones A et B.

La vendange se déroule en deux modes, le mode sélectif et le mode mixé. Au début, HM est en mode sélectif. Dans ce mode, la $a-trémie$ de HM contient uniquement des raisins A et la $b-trémie$ contient des raisins B mais peut également contenir des raisins A qui seront perdus pour la vendange de haute qualité. Dans le mode mixé, HM mélange les deux qualités de raisins. Dans ce mode, la $a-trémie$ et la $b-trémie$ contiennent des raisins A et B.

Il est possible que HM fasse la vidange de ses deux trémies dans les bennes situées autour de la parcelle même si ses deux trémies ne sont pas complètement pleines. En mode sélectif, si l'une des deux trémies est pleine alors HM doit vider les deux trémies. En mode mixé, HM peut attendre que les deux trémies soient pleines pour devoir vidanger.

Comme dans toute opération de récolte mécanisée de raisin, les trémies ne sont vidangées qu'à l'issue de la récolte sur tout un rang pour éviter de passer deux fois au même endroit. Il convient donc d'anticiper la quantité à récolter au rang suivant afin de décider si une vidange doit être effectuée à l'issue du rang courant.

5.3.2 Phénomène et problématique de latence

Pendant la récolte et comme expliqué dans la section précédente, le phénomène de *latence* peut se produire lorsque la machine se déplace d'une zone à l'autre. Analysons quelle incidence peut avoir une transition de zone de qualité de raisins sur la latence. Deux cas de figure peuvent (et doivent) être distingués.

Le premier cas est lorsque HM passe d'une zone A à une zone B (figure 5.3). Dans la zone A, les raisins sont dirigés vers la $a - trémie$. Dès que la machine pénètre dans la zone B, les raisins sont dirigés vers la $b - trémie$. Au moment où l'HM commence à pénétrer dans la zone B, il reste encore quelques raisins A dans la tête de la cueillette qui seront mélangés avec des raisins B. Pour être sûr que la $a - trémie$ ne contienne que la meilleure qualité, les raisins seront donc dirigés vers la $b - trémie$ dès le début de la transition entre les zones. Dans ce cas, il n'y a pas de latence. Dans ce sens de récolte, la quantité de raisins A perdus pour la récolte de haute qualité est minime.

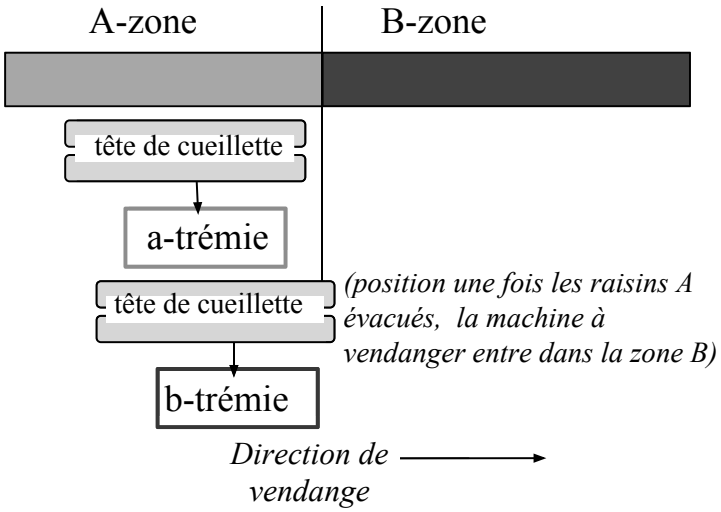


FIGURE 5.3 – Premier cas : passage d'une zone A à une zone B

Dans le deuxième cas (figure 5.4), les zones sont inversées, HM passe d'une zone B à une zone A. Dans la zone B, les raisins sont dirigés vers la $b - trémie$. Lorsque la machine finit la vendange de la zone B et pénètre dans la zone A, les raisins A peuvent se mélanger temporairement avec les raisins B restants dans la tête de la cueillette. Ces raisins doivent donc être triés vers la $b - trémie$ de sorte que la $a - trémie$ ne contienne pas de raisins B. Il faut donc attendre la fin de la transition, c'est-à-dire que la tête de la cueillette soit entièrement en zone A, pour pouvoir diriger les raisins vers la $a - trémie$. Cela provoque un temps de latence lors du passage d'une zone B à une zone A. Une fois ce temps écoulé, il n'y a que des

raisins A dans la tête de la cueillette et les raisins peuvent ainsi être dirigés vers la $a-trémie$.

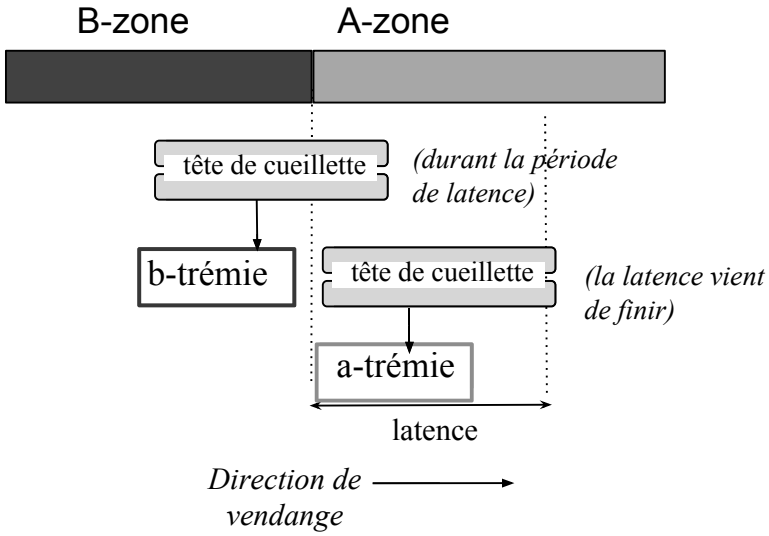


FIGURE 5.4 – Deuxième cas : passage d'une zone B à une zone A

Pendant le fonctionnement de la machine, la valeur du temps de latence lors du passage d'une zone B à une zone A est considérée comme un paramètre fixe (on considère que HM avance à vitesse fixe). Pour mémoire, lorsque la machine passe d'une zone A à une zone B, on a vu qu'il n'y avait pas de latence.

Cette asymétrie du temps de latence en fonction des transitions de zones rencontrées fait que les quantités de raisins A et B qui peuvent être récoltées dans un même rang dépendent du sens de déplacement de la machine dans ce rang. Prenons par exemple un rang avec une séquence de zones A-B-A-B si la machine récolte dans un sens donné. Si la machine récolte dans ce sens, il y a une seule transition B-A avec temps de latence. Par contre, si la machine récolte dans le sens inverse, il y a deux transitions B-A et donc deux fois plus de raisins A "perdus" dans la $b-trémie$. Les performances du processus de récolte peuvent donc dépendre de l'orientation de la machine de récolte lorsqu'elle parcourt les rangs.

5.3.3 Objectif de la vendange sélective

L'objectif agronomique du problème de vendange sélective est d'avoir au moins une quantité $Rmin$ de raisins A. Si ce seuil est atteint, et une fois que le contenu des deux trémies a été déversé dans les bennes, la machine passe en mode mixé (non sélectif) et place les raisins A et B dans n'importe quelle trémie. Si $Rmin$ a été atteint et que les trémies n'ont pas encore été vidées, la machine ne peut pas

passer du mode sélectif au mode mixé. On suppose que la machine commence les opérations en mode sélectif, puis passe au mode non sélectif une seule fois et y reste.

En raison des caractéristiques très spécifique de ce problème, parcourir et récolter une parcelle de façon classique, c'est-à-dire avec un motif régulier pratique pour les manoeuvres [3] ne conduira pas nécessairement à un parcours optimal. L'ordre et le sens de parcours des rangs vont influer sur l'instant de passage de la machine en mode mixé. De même, comme la capacité des trémies de la machine est limitée à $CmaxHopper$ et que les quantités de raisins à récolter différent d'un rang à un autre alors l'ordre de récolte des rangs influe sur le nombre de fois où HM vide ses trémies. Dans les deux cas cela pourra voir une influence sur la durée globale de la récolte. Ainsi, l'objectif du problème de vendange sélective en agriculture de précision et plus spécifiquement du DHP est de trouver l'ordre et l'orientation dans lesquels les rangs doivent être récoltés de manière à récolter au moins une quantité $Rmin$ de raisins A tout en minimisant la durée totale de la récolte.

5.4 Modélisation du problème et Propriétés à vérifier

Dans cette section, nous présentons une modélisation du problème de vendange sélective basée sur les automates temporisés de coût (*Priced Timed Automata*). Nous appelons le modèle conçu DHP_PTA

Le problème de vendange sélective a des similarités avec le problème classique de tournée de véhicule (Vehicle Rooting Problem, VRP) [78] qui définit une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Dans le cas d'une flotte limitée de véhicules, on peut le définir comme suit [78] : Soit un ensemble de n clients servis par une flotte de m véhicules. Chaque client doit être visité par exactement un véhicule. Chaque client i a une demande r_i, et la somme des demandes des clients affectés au véhicule k doit être inférieure à la capacité Q_k du véhicule. Tous les véhicules commencent et terminent leur trajet dans un seul dépôt. L'objectif du problème de VRP est de minimiser la somme des coûts de déplacement.

Dans notre cas, chaque rang à récolter peut être considéré comme un client et la flotte de véhicules se limite à une seule machine de récolte HM. Cependant, un rang ne pourra pas être considéré comme un sommet unique sur le graphe des clients du VRP, car le sens de parcours du rang a une influence sur le coût induit. HM peut aller plusieurs fois aux bennes situées quelque part au bord de la parcelle. En terminologie VRP, l'emplacement des bennes s'appelle le dépôt. Cela permet de

3. Par exemple, récolter le $i^{\text{ème}}$ rang dans un sens puis faire demi-tour et récolter le rang $i+2$ dans le sens inverse puis faire demi-tour et récolter le rang $i-1$ puis faire demi-tour et récolter le rang $i+1$

considérer le DHP comme assez proche d'un problème de routage à trajets multiples (en anglais *multi-trip VRP*) : l'itinéraire entier (en anglais *route*) étant composé de plusieurs trajets débutant au dépôt (emplacement des bennes), parcourant un sous-ensemble des rangs, et se terminant au dépôt. Il est supposé qu'il n'y a qu'un seul emplacement pour les bennes, avec 2 bennes, une pour les raisins A et une pour les raisins B & A.

Quatre points différencient cependant le DHP du VRP à trajets multiples :

1. Chaque rang est représenté par 2 extrémités. Dans un VRP, chaque client est représenté par un seul point.

2. Selon l'orientation de récolte du rang i, la demande r_i diffère. En effet, les quantités de raisins A et B collectées sont différentes suivant l'orientation de parcours du rang.

3. HM fonctionne en 2 modes (sélectif et mixé) selon le seuil *Rmin*. Dans un VRP, le véhicule fonctionne en un seul mode qui ne change pas.

4. HM a 2 trémies et par conséquent une capacité peut être définie pour chaque trémie. Dans un VRP, il y a une seule capacité par véhicule.

Nous pouvons maintenant définir toutes les informations relatives au DHP :

- Un champ de vigne est composé de n rangs.

- Chaque rang i est représenté par deux extrémités N_i et S_i. Toutes les extrémités N_i sont "du même côté" du champ, et la même règle s'applique aux extrémités S_i. Nous appelons *orientation directe*, l'orientation de N_i à S_i et *orientation inverse*, l'orientation de S_i à N_i.

- Une entrée du problème, que nous noterons Q_A_D (resp. Q_B_D), est donc le tableau des quantités de raisins A (resp. raisins B) contenues dans le rang à récolter dans l'orientation directe. Et Q_A_I, Q_B_I l'équivalent dans l'orientation inverse. Ces données du problème sont calculées à partir des cartes préalables de qualité de raisins de la parcelle étudiée, en prenant compte du phénomène de latence.

- L'espace entre les rangs et l'espace entre les pieds de vigne d'un même rang sont supposés constants. Le critère optimisé n'est pas la distance parcourue par HM mais le temps de parcours. Il convient donc de prendre en compte les temps de manœuvre lors des demi-tours, qui ne sont pas constants de part la manoeuvrabilité de la machine. Ainsi, pour passer d'une extrémité du rang i à l'extrémité du même côté du rang $i+1$, il faut plus de temps que pour passer du rang i au rang $i+2$ ou $i+3$.

- Une entrée du problème, que l'on notera DD, est donc la matrice des coûts. DD représente les durées de parcours entre toutes les extrémités de rang et entre chaque extrémité de rang et l'emplacement des bennes.
- Chaque trémie de la machine est limitée à une même capacité $CmaxHopper$.
- La machine doit récolter au moins la quantité $Rmin$ de raisins A.
- Si, au moment de la récolte, la a-tremie est pleine, la machine peut mettre les raisins A dans la b-tremie. Par conséquence, les raisins A peuvent être placés dans la a-trémie et la b-trémie, quelque soit le mode de récolte.
- Si, au moment de la récolte, la b-tremie est pleine, la machine ne peut pas mettre les raisins B dans la a-tremie lorsque le mode de récolte est sélectif.
- Pour garantir la qualité des raisins, le temps de latence est pris en considération dans le calcul des quantités A et B récoltées dans chaque rang et pour chaque orientation.
- La machine commence et finit la récolte à l'emplacement des bennes.
- La machine commence la récolte en mode sélectif.
- Si la machine est à l'emplacement des bennes, le prochain rang à récolter peut être récolté dans l'orientation directe ou dans l'orientation inverse.
- Si un rang est récolté dans l'orientation directe, le prochain rang récolté devra l'être dans l'orientation inverse, et vice versa. Dans la pratique, en effet, ne pas procéder ainsi conduirait à passer dans un rang sans le récolter, ou à contourner la parcelle, ce qui ne serait pas acceptable pour l'opérateur.

Nous appelons la dernière contrainte "contrainte d'orientation". Cette contrainte est une pratique agricole habituelle et efficace. Le modèle décrit ci-après est basé sur cette hypothèse. Cette contrainte permet également de réduire le nombre *d'états* à parcourir pendant la résolution du problème étant donné qu'elle diminue le nombre de possibilités de parcours.

Pour ce niveau, toutes les informations et les contraintes relatives au DHP ont été définies. La méthodologie de résolution du problème de vendange sélective que nous proposons est la suivante :

- Modéliser les contraintes du DHP sous la forme d'un réseau d'automates comprenant un automate modélisant les séquences de fonctionnement de la machine de récolte et autant d'automates que de rangs à récolter. Ces derniers permettent de modéliser si un rang a été ou non récolté et dans quel sens. La récolte d'un rang met à jour, par qualité de raisin, les quantités récoltées.

- Vérifier la propriété qui consiste à avoir récolté tous les rangs et qui permet de résoudre le problème du DHP.

L'outil de vérification UppAal-CORA permet la modélisation du réseau d'automates, la vérification de la propriété et il fournit en plus le coût minimal vérifiant cette dernière.

Dans la suite, les approches de N Briot dans la résolution du DHP sont décrites.

5.4.1 Approches de N. Briot pour la résolution du DHP

Dans les résultats présentés dans ce chapitre, nous comparons qualitativement les performances de notre approche avec celles rapportées dans les travaux de Nicolas Briot ([35, 36, 37, 38]). Il est donc important de rappeler brièvement ici les principes de ces travaux antérieurs qui sont basés sur des techniques d'optimisation sous contraintes (COP : Constraint Optimisation Problem).

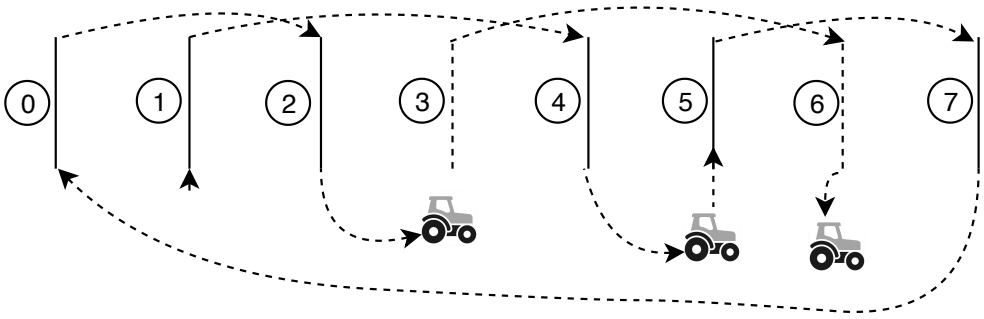
L'objectif de ce type d'approche est de trouver, si elle existe, la meilleure solution respectant les contraintes selon le critère d'optimisation. La programmation par contraintes se base sur deux principes : la modélisation du problème et sa résolution. La modélisation consiste à représenter le problème sous forme d'un problème de satisfaction de contraintes (un ensemble fini de variables, le domaine de chaque variable, et un ensemble fini de contraintes). La résolution consiste à trouver une ou des solutions à ce problème. Généralement, un solveur est utilisé pour la résolution. Un état de l'art sur les bases de la programmation sous contraintes (algorithmes, stratégies de recherche, etc.) est disponible dans [51].

Dans ([36], [37]), Nicolas Briot et ses co-auteurs ont proposé deux modèles.
Le premier a été appelé modèle *Step*. Il est basé sur 3 choix à effectuer à chaque étape de construction de l'itinéraire de la machine à vendanger :

1. quel est le rang suivant ?
2. dans quelle orientation le parcourir ?
3. faut-il vidanger les trémies après la récolte du rang courant ?

Un exemple illustratif d'une solution avec le modèle *Step* est donné dans la figure 5.5. La parcelle fait 8 rangs, les vidanges sont représentées sur le schéma par les trois véhicules et dans le tableau par les valeurs 1 sur la dernière ligne.

Selon les auteurs, les k premières étapes de l'itinéraire effectuent une récolte en mode sélectif. Avant l'étape k, les raisins B ne peuvent pas apparaître dans la a-trémie. A l'étape k, HM doit avoir récolté au moins la quantité Rmin de raisin A. Après l'étape k, les raisins A et B sont mélangés dans les deux trémies. Dans la première version du modèle *Step*, k est inclue en tant que variable dans le modèle et les résultats étaient très mauvais. L'aspect combinatoire du problème diminue si la valeur de k est fixée a priori. Par conséquent, chaque instance du problème initial



Étape	0	1	2	3	4	5	6	7
Rang	1	4	5	7	0	2	3	6
Orientation	0	1	0	1	0	1	0	1
Bennes ?	0	1	0	0	0	1	0	1

FIGURE 5.5 – Un exemple illustratif d'une solution avec le modèle *Step* [35]

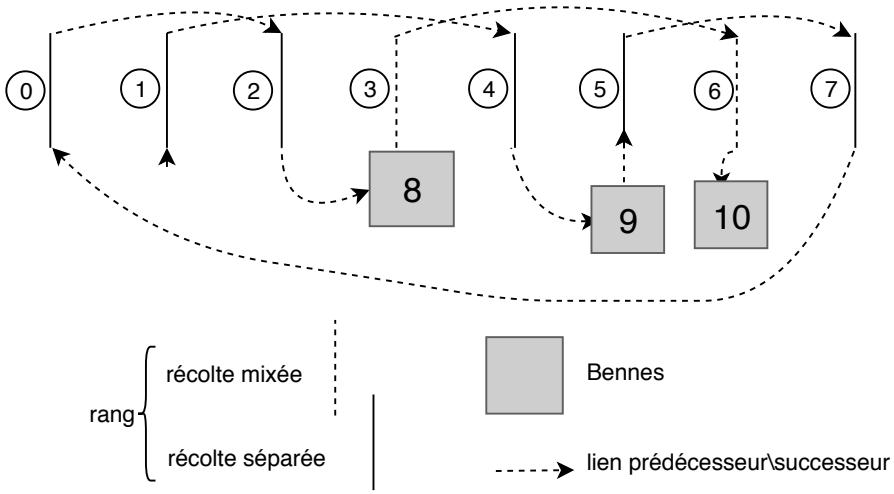
est remplacée par un ensemble de sous-instances avec différentes valeurs de k. Selon la capacité des trémies et la valeur de Rmin, la variable k peut avoir un nombre petit de valeurs possibles et comme ces instances sont indépendantes, elles peuvent être exécutées en parallèle puis la meilleure solution est récupérée.

Le second modèle a été appelé modèle *Successeur*. Il a été inspiré par [78] dans lequel un problème de routage de véhicule (VRP) a été résolu en utilisant la programmation par contraintes. Il utilise une contrainte appelée Circuit. Celle-ci cherche un ensemble de K circuits élémentaires disjoints appelés sous-tours dans un graphe. Si K = 1, la contrainte circuit vise à obtenir un seul circuit hamiltonien (un circuit qui passe par tous les sommets) et elle permet alors de respecter la contrainte d'élimination des sous-tours. Dans cette modélisation, un rang est décrit par 3 variables :

1. une décrit son prédécesseur (ou successeur),
2. une de type booléenne décrit son sens de traitement,
3. et une indique si la récolte est mixée ou pas.

La particularité du modèle *Successeur* est que les vidanges sont représentées par des sommets. Un exemple illustratif d'une solution avec le modèle *Successeur* est donné dans la figure 5.6. La parcelle fait 8 rangs, les vidanges sont représentées par les trois carrés rouges et par les valeurs 1 sur la dernière ligne du tableau.

Selon les auteurs, la résolution avec le modèle *Step* prend beaucoup de temps et est limité par la grande quantité de mémoire nécessaire. Il permet cependant



Sommet	0	1	2	3	4	5	6	7	8	9	10
Prédécesseur	7	10	0	8	1	9	3	5	2	4	6
Orientation	0	0	1	0	1	0	1	1			
Mixer ?	0	0	0	1	0	0	1	0	0	0	1

FIGURE 5.6 – Un exemple illustratif d'une solution avec le modèle *Successeur* [35]

de résoudre des instances comprenant 12 rangs. Les performances obtenues avec le modèle *Successeur* sont meilleures. Avec ce modèle, il est possible d'obtenir le temps de récolte optimal pour 14 rangs.

Dans la suite de ce chapitre, nous présenterons notre méthode de modélisation et de résolution du DHP, qui met en œuvre l'outil UppAal-CORA et ne comprend pas d'heuristique dédiée. Elle sera comparée qualitativement aux méthodes publiées par Nicolas Briot et ses co-auteurs.

5.4.2 Description de la matrice des coûts

Nous rappelons que la matrice de coût représente les durées de parcours entre toutes les extrémités de rang et entre chaque extrémité de rang et l'emplacement des bennes. Dans cette section, nous expliquons comment la matrice de coût est construite en l'illustrant sur un exemple comportant 3 rangs de vigne.

Comme indiqué plus haut, chaque rang i dans le champ est représenté par deux extrémités N_i et S_i et la matrice des coûts représente le temps requis pour passer d'une extrémité d'un rang à une autre ou d'une extrémité d'un rang à l'emplacement des bennes. Le graphe de la figure 5.7 illustre les données du problème de vendange sélective pour un exemple comportant 3 rangs de vigne et 1 emplacement de bennes.

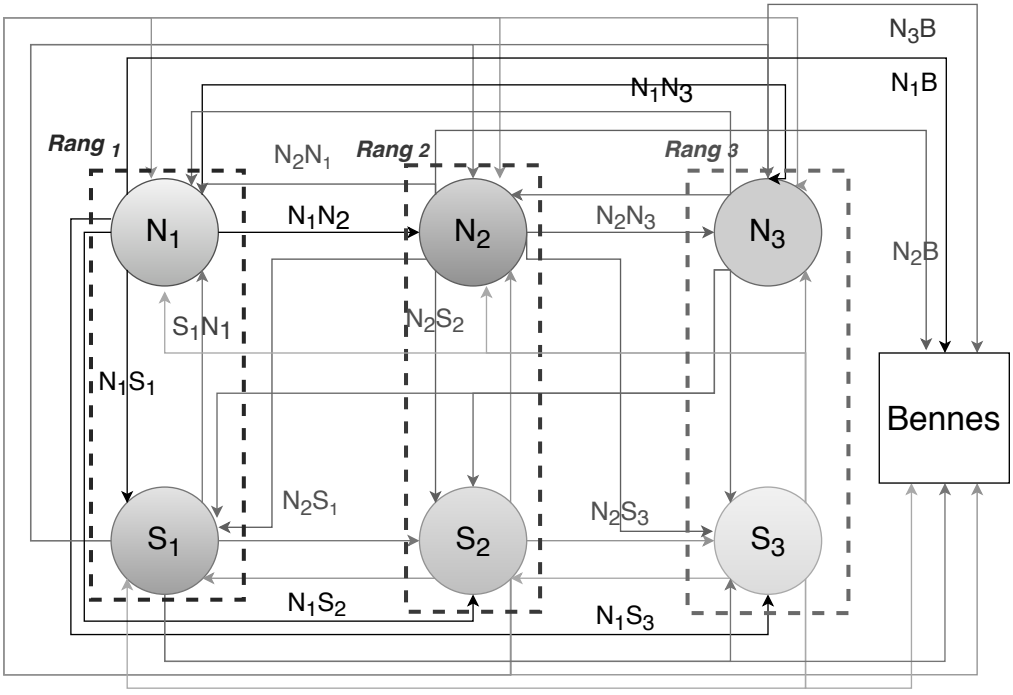


FIGURE 5.7 – Données du problème de vendange sélective : un cas avec 3 rangs et 1 emplacement de bennes

Dans la figure 5.7, une couleur est attribuée à chaque extrémité du rang et aux arcs sortants de cette extrémité. Par exemple, la couleur noire est affectée à l'extrémité N_1. À partir de N_1, HM peut visiter N_2, N_3, S_1, S_2, S_3 et les bennes (B). En raison de la contrainte d'orientation (deux rangs récoltés l'un après l'autre doivent être récoltés dans des orientations opposées), il s'ensuit que de N_1, HM n'ira pas en S_2 et S_3. Donc N_1S_2 (temps de trajet de N_1 à S_2) et N_1S_3 (temps de trajet de N_1 à S_3) ne sont pas considérés (symbole $-$ dans la matrice). De plus, BB, S_iS_i et N_iN_i sont bien sûr égaux à 0 car HM reste au même endroit.

Logiquement, HM étant une grosse machine, elle effectue beaucoup de manoeuvres lorsqu'elle passe du rang i au rang i + 1, et beaucoup moins lorsqu'elle passe du rang i au rang i + 3. Par exemple pour notre machine, des experts ont estimé qu'une manœuvre pour passer du rang courant au rang suivant demande 10 secondes. En l'absence de manœuvre le temps nécessaire pour atteindre le rang i+3 depuis le rang courant a été estimé à 6 secondes. Cette information est utilisée dans le calcul de la matrice des coûts.

Dans la matrice, les lignes représentent les points de départ qui peuvent être ou bien l'emplacement des bennes ou bien une extrémité du rang, les colonnes représentent les points d'arrivée qui peuvent être également ou bien l'emplacement

des bennes ou bien une extrémité du rang. Chacune des composantes représente le temps de parcours du point de départ au point d'arrivée. En se basant sur la figure 5.7, la matrice des coûts DD est alors la suivante :

$$DD = \begin{array}{c} \\ De\ Bennes \\ De\ N_1 \\ De\ S_1 \\ De\ N_2 \\ De\ S_2 \\ De\ N_3 \\ De\ S_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} Bennes & N_1 & S_1 & N_2 & S_2 & N_3 & S_3 \end{array} \\ \begin{pmatrix} 0 & BN_1 & BS_1 & BN_2 & BS_2 & BN_3 & BS_3 \\ N_1B & 0 & N_1S_1 & N_1N_2 & - & N_1N_3 & - \\ S_1B & S_1N_1 & 0 & - & S_1S_2 & - & S_1S_3 \\ N_2B & N_2N_1 & - & 0 & N_2S_2 & N_2N_3 & - \\ S_2B & - & S_2S_1 & S_2N_2 & 0 & - & S_2S_3 \\ N_3B & N_3N_1 & - & N_3N_2 & - & 0 & N_3S_3 \\ S_3B & - & S_3S_1 & - & S_3S_2 & S_3N_3 & 0 \end{pmatrix} \end{array}$$

Évidemment, le temps de parcours XY est égal au temps de parcours YX avec X et Y pouvant représenter une extrémité de rang ou l'emplacement des bennes. La matrice est donc symétrique et elle peut alors être réduite à :

$$dd = \begin{array}{c} \\ De\ Bennes \\ De\ Rang_1 \\ De\ Rang_2 \\ De\ Rang_3 \end{array} \begin{array}{c} \begin{array}{cccc} Bennes & Rang_1 & Rang_2 & Rang_3 \end{array} \\ \begin{pmatrix} 0 & BN_1 & BN_2 & BN_3 \\ S_1B & N_1S_1 & N_1N_2 & N_1N_3 \\ S_2B & S_2S_1 & N_2S_2 & N_2N_3 \\ S_3B & S_3S_1 & S_3S_2 & N_3S_3 \end{pmatrix} \end{array} \tag{5.1}$$

Dans cette matrice réduite dd, la partie triangulaire supérieure représente les temps de parcours de l'emplacement des bennes aux extrémités N_i et les temps de parcours entre les extrémités N_i. La colonne diagonale représente le temps de récolte de chaque rang. La partie triangulaire inférieure représente les temps de parcours depuis les extrémités S_i jusqu'à l'emplacement des bennes et les temps de parcours entre les extrémités S_i.

5.4.3 Modélisation du problème DHP

Le modèle DHP_PTA est un réseau de $n+1$ automates : un automate de la machine à vendanger et n automates de rang. Chaque automate de rang est instancié avec un numéro d'index du rang. Les variables ont été définies pour être cohérentes avec [36]. Le modèle est implémenté en ATC avec UppAal-CORA. Dans la suite, les détails du modèle DHP_PTA sont fournis.

5.4.3.1 Déclaration des constantes et des variables

Cette section définit les constantes et les variables déclarées dans le modèle.

A) Déclaration des constantes :

$CmaxHopper$: est la capacité maximale de chaque trémie.

Rmin : est la quantité minimale de raisins A qui doit être récoltée. Elle est fournie en pourcentage de la quantité totale de raisins A identifiés sur la parcelle.

Q_A_D[*i*] (*resp. Q_B_D*[*i*]) : est la quantité de raisins A (respectivement de raisins B) qui sera collectée dans le rang (i+1) si ce rang est récolté selon l'orientation directe.
Le décalage d'indice s'explique par le fait que, dans la matrice *dd* l'indice du premier rang est 1 car l'indice 0 est réservé à l'emplacement des bennes. Or on ne collecte le raisin que dans les rangs, donc le premier indice du tableau *Q_A_D* est 0 et *Q_A_D*[0] représente la quantité de raisins A qui sera collectée dans le rang 1.

Q_A_I[*i*](*resp. Q_B_I*[*i*]) : est la quantité des raisins A (respectivement de raisins B) qui sera collectée dans le rang (i+1) si ce rang est récolté selon l'orientation inverse.

dd : est la matrice réduite des coûts.

DumpTime : est le temps nécessaire pour vider les trémies dans les bennes.

B) Déclaration des variables :

$Mixed$: est une variable booléenne qui représente le mode de récolte. Si $Mixed = 0$, le mode de récolte est sélectif, sinon $Mixed = 1$.

T : est une variable entière qui représente le temps de déplacement de HM. Elle est calculée de deux façons : 1) dans l'automate d'un rang (figure 5.8), elle prend en compte le temps que la machine met pour faire un déplacement vers le rang suivant et le récolter ; 2) dans l'automate de la machine (figure 5.9), elle prend en compte le temps pour se rendre à l'emplacement des bennes et vider la récolte dans les bennes.

Location : est une variable entière qui représente l'emplacement actuel de HM. Si $Location = 0$, la machine est à l'emplacement des bennes. Sinon, si $Location = i$ avec $i \in [1 : n]$, alors la machine est au rang i.

DumpNumber : est une variable entière qui représente le nombre de vidanges effectuées depuis le début de la récolte.

$Ori[i]$: est une variable booléenne qui mémorise l'orientation de récolte d'un rang. Si $Ori[i] = 1$, le rang $(i + 1)$ est récolté dans l'orientation directe, sinon il est récolté dans l'orientation inverse.

$Mix[i]$: est une variable booléenne qui mémorise le mode de récolte d'un rang. Si $Mix[i] = 0$, le rang $(i+1)$ est récolté en mode sélectif sinon il est récolté en mode mixé (non sélectif).

U_A (*resp. U_B*) : représente la quantité courante de raisins A (resp. raisins B) dans la a-trémie (resp. b-trémie).

Q_T_A (*resp. Q_T_B*) : représente la quantité totale de raisins A (resp. raisins B) récoltée dans les bennes depuis le début de la récolte jusqu'à la dernière vidange.

c est une horloge locale pour le modèle de la machine.

A ce stade, toutes les constantes et les variables nécessaires à la compréhension des

automates ont été définies. Dans la suite, les automates du modèle DHP_PTA sont décrits.

5.4.3.2 Description de l'automate de rang

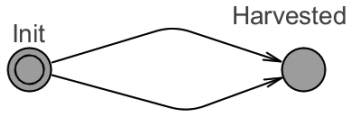
Dans l'automate du modèle de rang représenté dans la figure 5.8, il y a deux parties similaires qui correspondent à chaque orientation potentielle.

À l'état *Init*, le rang i n'est pas récolté. Il peut être récolté selon l'orientation directe (transition supérieure) ou l'orientation inverse (transition inférieure). Si la valeur de la condition de la transition est vraie ((1) ou (2) en vert dans la figure 5.8), alors le rang passe à l'état *Harvested* (récolté).

```
((Mixed==0)? (U_B + Q_B_D[i-1]<=CmaxHopper
&& U_A + Q_A_D[i-1] + U_B + Q_B_D[i-1]<= 2*CmaxHopper)          (1)
 : (U_A + Q_A_D[i-1] +  U_B + Q_B_D[i-1]<= 2*CmaxHopper))
&&(Location==0 || Ori[Location-1]==0)

toRow!

Ori[i-1]=1,
T= dd[Location][i] + dd[i][i],
Location = i,
Mix[i-1] = Mixed,
(U_A + Q_A_D[i-1] <= CmaxHopper)? (U_A += Q_A_D[i-1]):
(U_B = U_B+ U_A + Q_A_D[i-1]- CmaxHopper) && (U_A = CmaxHopper),
(U_B + Q_B_D[i-1] <= CmaxHopper)? (U_B += Q_B_D[i-1]):
(U_A = U_A+ U_B + Q_B_D[i-1]- CmaxHopper) && (U_B = CmaxHopper)
```




```
((Mixed==0)? (U_B + Q_B_I[i-1] <= CmaxHopper
&& U_A + Q_A_I[i-1] + U_B + Q_B_I[i-1]<= 2*CmaxHopper)          (2)
 : (U_A + Q_A_I[i-1] + U_B + Q_B_I[i-1]<= 2*CmaxHopper))
&& (Location==0 || Ori[Location-1]==1)

toRow!

Ori[i-1]=0,
T= dd[i][Location] + dd[i][i],
Location = i,
Mix[i-1] = Mixed,
(U_A + Q_A_I[i-1] <= CmaxHopper)? (U_A += Q_A_I[i-1]):
(U_B = U_B+ U_A + Q_A_I[i-1]- CmaxHopper) && (U_A = CmaxHopper),
(U_B + Q_B_I[i-1] <= CmaxHopper)? (U_B += Q_B_I[i-1]):
(U_A = U_A+ U_B + Q_B_I[i-1]- CmaxHopper) && (U_B = CmaxHopper)
```

FIGURE 5.8 – Modèle de l'automate du rang pour chaque rang $i \in [1 : n]$

Examinons tout d'abord la transition supérieure, qui a la condition (1) [4]. Cette condition correspond au cas ou le rang i sera récolté dans une orientation directe. Il s'ensuit que soit le rang précédent a été récolté dans l'orientation inverse ($Ori[Location-1] == 0$), soit l'emplacement de la machine avant la transition était l'emplacement des bennes ($Location == 0$). L'automate vérifie également dans la condition (1), en fonction de la valeur de la variable $Mixed$, s'il reste suffisamment de volume disponible dans les trémies pour récolter les raisins du rang i :

- Si $Mixed == 0$, le mode de récolte est sélectif. La quantité $U\_B + Q\_B\_D[i-1]$ (raisins déjà dans la b-trémie + raisins B à collecter) ne doit pas dépasser $CmaxHopper$. En outre, la quantité $U\_A + U\_B + Q\_A\_D[i-1] + Q\_B\_D[i-1]$ (raisins déjà dans les deux trémies + raisins A et B à récolter) ne doit pas dépasser $2 \times CmaxHopper$ (car les raisins A peuvent être placés dans la a-trémie et la b-trémie, quelque soit le mode de récolte).
- Si $Mixed == 1$, le mode de récolte est mixé (pas sélectif). L'automate du rang vérifie alors que la quantité $U\_A + U\_B + Q\_A\_D[i-1] + Q\_B\_D[i-1]$ ne dépasse pas $2 \times CmaxHopper$.

Si la valeur de la condition de la transition est vraie, l'automate du rang envoie un signal de synchronisation *toRow*! à l'automate de la machine et met à jour les variables $Ori[i-1]$, $Mix[i-1]$, $Location$, U_A, U_B et T (instructions en bleu).

$Ori[i-1]$ est mise à jour à 1 si le $i^{ème}$ rang est récolté selon une orientation directe, sinon elle est mise à jour à 0. $Mix[i-1]$ est mise à jour à la valeur de $Mixed$. $Location$ est mise à jour à i.

Enfin, T est mise à jour à la somme du temps de parcours entre rangs et du temps de récolte dans le rang. Le temps de récolte est $dd[i][i]$. Le temps de parcours dépend de l'orientation avec laquelle le rang sera récolté. Si le rang doit être récolté selon l'orientation directe, alors le temps de parcours est égal à $dd[Location][i]$ sinon il est égal à $dd[i][Location]$. Ceci résulte de la définition de la matrice des coûts dd (équation 5.1).

Enfin, l'automate met à jour les quantités de raison. Ce calcul de U_A et U_B nécessite quelques explications.

- U_A représente la quantité de raisins dans la a-trémie à la fin du rang i.
  - Si $U\_A + Q\_A\_D[i-1]$ est inférieure à $CmaxHopper$, alors la variable U_A est mise à jour à la valeur $U\_A + Q\_A\_D[i-1]$.
  - Sinon, cela signifie que la quantité de raisins A dépasse la capacité de la a-trémie : U_A est mise à jour à la valeur de $CmaxHopper$, et le reste de raisins A ($U\_A + Q\_A\_D[i-1] - CmaxHopper$) est placé dans la b-trémie. U_B est donc mise à jour en conséquence.

4. La transition inférieure est construite de manière similaire et ne nécessitera donc pas d'explications supplémentaires.

- U_B représente la quantité de raisins dans la b-trémie à la fin du rang i.
  - Si $U\_B + Q\_B\_D[i-1]$ est inférieur à $CmaxHopper$, alors U_B est mise à jour à la valeur $U\_B + Q\_B\_D[i]$.
  - Sinon, U_B est mise à jour à la valeur de $CmaxHopper$ et le reste des raisins B ($U\_B + Q\_B\_D[i] - CmaxHopper$) est mise dans la a-trémie. Donc, U_A est mise à jour avec ce reste de raisins B. Il est à noter que ce dernier cas ne se produira que dans le mode de récolte mixte (non sélectif). En effet, le cas $U\_B + Q\_B\_D[i] > CmaxHopper$ ne peut se produire qu'en mode mixte. Pour cela, la garde des transitions vérifie bien que si $Mixed == 0$, alors $U\_B + Q\_B\_D[i] \leq CmaxHopper$.

5.4.3.3 Description de l'automate de la machine de récolte

L'automate de la machine de récolte est écrit par la figure 5.9. Il comporte 4 états : *"Bin"*, *"Wait _Harvesting"*, *"Next_Row"* et *"Wait _Dumping"*.

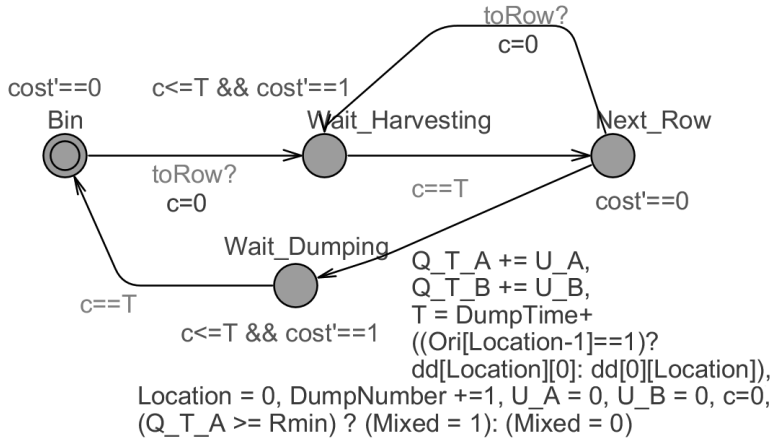


FIGURE 5.9 – Automate de la machine de récolte HM

Au moment de l'initialisation, l'emplacement de départ de HM est l'emplacement des bennes (état *Bin*). Dès que l'automate de la machine reçoit un signal de synchronisation *toRow*? à partir d'un automate de rang, ce qui correspond à une commande de récolte d'un rang, l'horloge locale c est réinitialisée et la transition vers l'état suivant *Wait _Harvesting* est franchie.

A l'état *Wait _Harvesting*, HM attend pendant la durée de récolte du rang, représentée par la variable T qui a été calculée par l'automate de rang au moment de l'envoie du signal *toRow*?. Une fois ce temps écoulé, la machine a alors fini de récolter le rang et l'automate de HM passe à l'état *Next_Row*.

À ce stade, HM est à la fin d'un rang. Deux cas sont possibles. Dans le premier cas, l'automate reçoit de nouveau un signal de synchronisation *toRow*? d'un automate de rang, qui le fait retourner à l'état *Wait _Harvesting*. Dans le deuxième cas, la vidange des trémies de la machine dans les bennes doit être effectuée, la transition de l'état *Next _Row* à l'état *Wait _Dumping* sera donc franchie et l'automate HM mettra à jour les variables Q_T_A, Q_T_B, $Mixed$, $Location$, $DumpNumber$, U_A, U_B, c et T comme suit :

Q_T_A (resp. Q_T_B) est mise à jour à la valeur $Q\_T\_A + U\_A$ (resp. $Q\_T\_B + U\_B$) et U_A (resp. U_B) est réinitialisée. $Location$ est mise à jour à 0 (emplacement des bennes). $DumpNumber$ est incrémentée. Si $Rmin$ est atteint ($Q\_T\_A >= Rmin$), $Mixed$ est mise à 1 pour passer en mode mixé. Sinon $Mixed$ reste à 0 et la récolte sélective continue.

Sur cette transition, T représente le temps de vidange des trémies ($DumpTime$) plus le temps de parcours entre le rang et l'emplacement des bennes. Ce temps de parcours dépend de l'orientation avec laquelle le rang précédent ($Location - 1$) a été récolté. S'il a été récolté selon l'orientation directe ($Ori[Location - 1] == 1$), le temps de parcours est $dd[Location][0]$, sinon il est $dd[0][Location]$.

A l'état *Wait _Dumping*, l'automate de HM attend les unités de temps T. Lorsque ce temps s'est écoulé, l'automate HM repasse à l'état initial *Bin*.

5.4.4 Vérification du modèle et calcul de la séquence de contrôle à l'aide d'UppAal-Cora

De nombreuses propriétés peuvent être vérifiées sur le modèle qui vient d'être construit pour en vérifier la cohérence et le comportement. Certaines d'entre-elles sont présentées dans la suite et formulées en logique temporelle CTL.

1. Est-ce-que tous les rangs peuvent être récoltés ?
   PRS1 : $E <> Row1.Harvested \wedge Row2.Harvested \wedge .. Rown.Harvested$

2. Vérifier des propriétés de sûreté relatives aux bornes maximales des variables :

   - Le nombre de vidanges est-il toujours inférieur au nombre de rangs ?
     PRS2 : $A[\,]\ DumpNumber \leq nbRow$
   - La quantité de raisins dans les trémies est-elle toujours inférieure à la capacité maximale des trémies ?
     PRS3 : $A[\,]\ (U\_A \leq CmaxHopper$ et $U\_B \leq CmaxHopper)$

3. Tous les rangs peuvent être récoltés et Q_T_A est supérieure à $Rmin$ et HM commence et finit son travail de/à l'emplacement des bennes.
   PRS4 : $E <> (Row1.Harvested$ et $Row2.Harvested$ et ... $Rown.Harvested$ et $HM.Bin$ et $Q\_T\_A >= Rmin)$

C'est cette dernière propriété qui permet d'adresser le problème du DHP que nous cherchons à résoudre.

Nous verrons dans la section suivante les résultats obtenus de la mise en oeuvre de notre méthodologie DHP sur un exemple réel.

5.5 Résultats de l'application de la méthode à une parcelle réelle

Dans cette section, nous présentons d'abord les caractéristiques de la parcelle à récolter et de la machine de récolte. Ensuite, nous discutons les performances du modèle DHP_PTA pour différentes instances de 7 à 12 rangs extraits du champ de vigne réel.

5.5.1 Caractéristiques de la parcelle et de la machine

La parcelle, appelée Grenache24, est un vignoble, située à l'INRA de Pech-Rouge dans le sud de la France, à proximité de Gruissan, dans l'Aude. La figure 5.10 propose une vue aérienne du champ étudié.

FIGURE 5.10 – Le vignoble grenache avec les deux qualités de raisins : qualité A en rouge et qualité B en vert

Les caractéristiques de la parcelle sont les suivantes : elle est composée de 24 rangs, l'inter-rang dans le champ est égal à 2,5 m, la distance entre deux ceps est

égale à 1 m et les 12 premiers rangs ont une longueur approximative de 189 m. La zone rouge correspond aux raisins de qualité A et les 3 zones vertes délimitent les zones des raisins de qualité B. Les quantités de raisins A et B dans chaque rang sont calculées comme suit : le rendement sur les zones de qualité A est 1,5 kg/cep et le rendement sur les zones de qualité B est 2,5 kg/cep. Le coefficient de conversion de kg de raisins en volume (en litres) stocké dans la trémie est 0,9 (1 litre = 0,9 kg). Les données de qualités de raisin et de calcul ont été fournies par Montpellier SupAgro. Les données ont été collectées pour $n_{max} = 24$ rangs. L'emplacement des bennes est supposé être près de l'extrémité N_2.

Les caractéristiques de la machine à vendanger sont les suivantes. La vitesse d'avancement en récolte dans les rangs est supposée être constante, fixée à une valeur de 4 $km.h^{-1} = 1.1\ m.s^{-1}$. La vitesse d'avancement en se déplaçant sans récolter, c'est-à-dire d'un rang à l'autre ou entre l'extrémité d'un rang et l'emplacement des bennes, est aussi supposée fixe mais à 9 $km.h^{-1} = 2.5\ m.s^{-1}$

Pour calculer les temps de parcours entre les extrémités des rangs, nous avons besoin de prendre en considération le temps de manoeuvre. Il est défini comme suit :

- pour passer d'un rang au suivant : 10 secondes,
- pour passer d'un rang i à un rang $i + 2$: 8 secondes,
- pour passer d'un rang i à un rang $i + 3$: 6 secondes,
- pour passer d'un rang i à un rang $i + 4$: 5 secondes,
- pour passer d'un rang i à un rang $i + k$ $(k > 4)$: 3 secondes (décélération/accélération) + d/v avec (d : distance entre les rangs, v : vitesse de transport hors rang).

Toutes ces informations permettent de calculer les quantités de raisins A et B dans chaque rang selon l'orientation en prenant en compte la latence (figure 5.3, figure 5.4) et de calculer la matrice des coûts (équation 5.1 section 5.4.2). Dans notre cas, la matrice des coûts dd pour les 24 rangs est donnée dans le tableau 5.1.

	B	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}	R_{18}	R_{19}	R_{20}	R_{21}	R_{22}	R_{23}	R_{24}
De B	0	16	4	4	4	5	6	7	8	8	10	10	11	12	13	15	16	18	20	21	24	25	26	27	28
De R_1	78	62	12	13	14	15	16	17	18	19	20	21	22	23	24	26	28	30	32	34	37	38	39	41	42
De R_2	78	12	74	10	8	6	4	5	6	7	8	9	10	11	12	15	16	18	20	22	25	26	27	29	30
De R_3	78	13	10	74	10	8	6	4	5	6	7	8	9	10	11	14	15	17	19	21	24	25	26	28	29
De R_4	79	14	8	10	75	10	8	6	4	5	6	7	8	9	10	12	14	16	18	20	23	24	25	27	28
De R_5	80	15	6	8	10	75	10	8	6	4	5	6	7	8	9	11	13	15	17	19	22	23	24	26	27
De R_6	80	16	4	6	8	10	74	10	8	6	4	5	6	7	8	10	12	14	16	18	21	22	23	25	26
De R_7	81	17	5	4	6	8	10	74	10	8	6	4	5	6	7	9	11	13	15	17	20	21	22	24	25
De R_8	82	18	6	5	4	6	8	10	74	10	8	6	4	5	6	8	10	12	14	16	19	20	21	23	24
De R_9	83	19	7	6	5	4	6	8	10	74	10	8	6	4	5	7	9	11	13	15	18	19	20	22	23
De R_{10}	84	20	8	7	6	5	4	6	8	10	74	10	8	6	4	6	8	9	12	14	16	18	19	20	21
De R_{11}	85	21	9	8	7	6	5	4	6	8	10	75	10	8	6	5	7	9	11	13	16	17	18	19	21
De R_{12}	86	22	10	9	8	7	6	5	4	6	8	10	75	10	8	6	6	8	10	12	15	16	17	19	20
De R_{13}	77	23	11	10	9	8	7	6	5	4	6	8	10	65	10	8	6	7	9	11	14	15	16	18	19
De R_{14}	76	24	12	11	10	9	8	7	6	5	4	6	8	10	63	10	8	6	8	10	13	14	15	17	18
De R_{15}	74	26	15	14	12	11	10	9	8	7	6	5	6	8	10	59	10	8	6	7	10	12	13	14	15
De R_{16}	74	28	16	15	14	13	12	11	10	9	8	7	6	6	8	10	58	10	8	6	9	10	11	13	14
De R_{17}	72	30	18	17	16	15	14	13	12	11	9	9	8	7	6	8	10	54	10	8	7	8	10	11	12
De R_{18}	69	32	20	19	18	17	16	15	14	13	12	11	10	9	8	6	8	10	50	10	8	6	7	8	9
De R_{19}	67	34	22	21	20	19	18	17	16	15	14	13	12	11	10	7	6	8	10	46	10	8	6	7	8
De R_{20}	61	37	25	24	23	22	21	20	19	18	16	16	15	14	13	10	9	7	8	10	37	10	8	6	5
De R_{21}	59	38	26	25	24	23	22	21	20	19	18	17	16	15	14	12	10	8	6	8	10	34	10	8	6
De R_{22}	58	39	27	26	25	24	23	22	21	20	19	18	17	16	15	13	11	10	7	6	8	10	32	10	8
De R_{23}	41	41	29	28	27	26	25	24	23	22	20	19	19	18	17	14	13	11	8	7	6	8	10	14	10
De R_{24}	34	42	30	29	28	27	26	25	24	23	21	21	20	19	18	15	14	12	9	8	5	6	8	10	6

TABLE 5.1 – Matrice des coûts de notre parcelle exemple

5.5.2 Résolution du problème de DHP

Nous cherchons maintenant à résoudre ce problème de DHP en nous appuyant sur le modèle DHP_PTA que nous avons construit. Pour ce faire la propriété d'atteignabilité PRS4 est formulée et vérifiée dans UppAal-CORA en sélectionnant l'ordre de recherche Best-first. Cela veut dire que la recherche s'effectue au sein du graphe en choisissant d'explorer à chaque étape le nœud le plus prometteur et la résolution permet de fournir le coût optimal (le temps minimum de la récolte).

Afin d'étudier empiriquement la complexité de la résolution du DHP avec le modèle DHP_PTA, la requête a d'abord été appliquée sans utiliser la fonctionnalité d'optimisation par *remaining* de UppAal-CORA, que l'on expliquera par la suite.

La table 5.2 et la figure 5.11 présentent les résultats de cette analyse en indiquant pour chaque instance le nombre d'états explorés pendant la résolution et le coût optimal obtenu. Si le nombre de rangs dépasse 9 rangs, un problème d'explosion combinatoire se produit et la vérification devient impossible. Dans la table 5.2, la première colonne désigne les instances du problème DHP soumis à UppAal-CORA. Le codage $nCiRi$ est choisi pour désigner les instances avec n le nombre de rangs ($n \in [7, 12]$), Ci la capacité maximale de la trémie en litres (égale à $C1$ ou $C2$, avec $C1 = 1000l$ et $C2 = 2000l$), et Ri qui représente la façon de calculer $Rmin$ ($Rmin = R1 \times QAtot$ ou $Rmin = R2 \times QAtot$, avec $QAtot$ la somme des valeurs des quantités de raisin A dans l'ensemble des rangs, $R1 = 50\%$ et $R2 = 70\%$). La deuxième colonne désigne le nombre d'états explorés pendant la résolution de cette instance et la troisième colonne désigne le coût optimal obtenu.

TABLE 5.2 – Résultats expérimentaux sans remaining

Instance	Nombre d'états	Coût	Instance	Nombre d'états	Coût
7C1R1	264113	655	9C1R1	9295636	827
7C1R2	199744	655	9C1R2	4297934	827
7C2R1	436221	655	9C2R1	18497531	809
7C2R2	337626	655	9C2R2	12763043	809
8C1R1	945767	669	8C1R2	726570	676
8C2R1	2077252	668	8C2R2	1670658	668

Ces résultats sont issus de l'application de la commande suivante :

$$./memtime\ ./verifyta\ -u\ -t3\ -o3\ -f\ tracefile\ model.xml\ query.q$$

Comme mentionné dans le chapitre 2, la fonction *memtime* permet la mesure de la mémoire et du temps utilisé lors de l'exécution d'une commande. La fonction *verifyta* permet de vérifier la propriété décrite dans le fichier *query.q* sur le modèle DHP_PTA décrit dans le fichier *model.xml*. Les options de vérification utilisées sont -u, -t3, -o3 et -f. L'option -u permet de donner le nombre d'états stockés et le

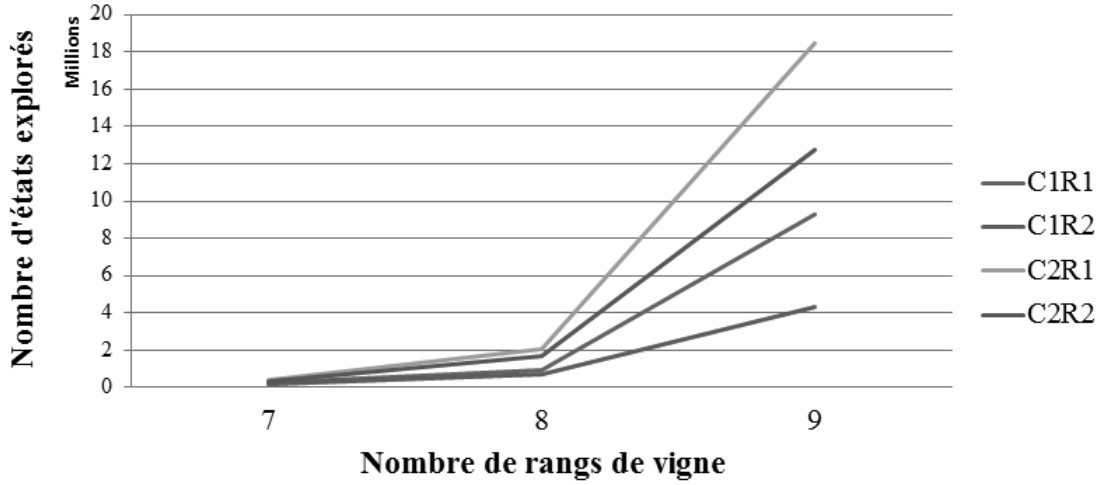


FIGURE 5.11 – Analyse quantitative du nombre d'états explorés en fonction du nombre de rangs récoltés

nombre d'états explorés. L'option -t3 génère la trace avec le meilleur coût. L'option -o3 sélectionne l'ordre de recherche optimal first et l'option -f ajoute un préfixe au nom de la trace. Le fichier *tracefile* est le nom du fichier trace, le fichier *model.xml* peut avoir le nom $nCiRi.xml$ avec $nCiRi$ représentant le codage défini précédemment. Le fichier *query.q* contient la propriété qui permet de résoudre le problème du DHP qui est la propriété PRS4.

5.5.2.1 Étude du gain obtenu

Pour évaluer l'efficacité de notre méthode et estimer le gain obtenu avec l'approche CORA, il est intéressant de faire une comparaison sur le coût (temps de récolte) avec une machine qui parcourrait et récolterait la parcelle de façon classique c'est-à-dire avec un motif régulier pratique pour les manoeuvres. Par exemple, récolter le $i^{\text{ème}}$ rang dans un sens puis faire demi-tour et récolter le rang $i+2$ dans le sens inverse puis aller à la benne. Il suffit de répéter ce motif jusqu'à la fin de la récolte de toute la parcelle. Le tableau 5.3 présente une comparaison des coûts entre une approche classique et une approche CORA. La première colonne désigne les instances (même codage qu'auparavant). La colonne nommée "Coût classique" donne le coût si la machine suit le motif régulier et la colonne nommée "Coût optimal" donne le coût optimal trouvé par notre approche. La colonne nommée "Gain" représente le gain obtenu grâce à notre approche.

D'après les résultats du tableau 5.3, notre méthode permet d'économiser entre 6 et 9 % du coût par rapport à une approche classique. Il est surtout important de noter que ce gain n'est pas constant, et qu'il augmente avec le nombre de rangs. Il est sera ainsi possible d'atteindre des gains non négligeable, d'autant plus si on

TABLE 5.3 – Comparaison du coût (temps de récolte) entre une approche classique et une approche CORA

Instance	Coût classique	Coût optimal	Gain	Instance	Coût classique	Coût optimal	Gain
7C1R1	703	655	6,82%	9C1R1	888	827	6,87%
7C1R2	703	655	6,82%	9C1R2	888	827	6,87%
7C2R1	703	655	6,82%	9C2R1	888	809	8,89%
7C2R2	703	655	6,82%	9C2R2	888	809	8,89%
8C1R1	713	669	6,17%	8C1R2	713	676	5,19%
8C2R1	713	668	6,31 %	8C2R2	713	668	6,31 %

arrive à maîtriser la complexité du problème.

5.5.2.2 Analyse de la complexité

Nous allons maintenant analyser la sensibilité du problème de DHP en fonction des paramètres qui influencent sa complexité de façon à chercher à mieux maîtriser l'explosion combinatoire.

Influence du nombre de rangs

La forte augmentation de la complexité en fonction du nombre des rangs, que l'on peut observée figure 5.11, était attendue, dans la mesure où le choix de l'ordre de parcours des n rangs génère une combinatoire de type $n!$. La combinatoire est en réalité supérieure à $n!$ dans la mesure où, à chaque fin de rang, il faut ajouter la décision d'aller vidanger ou non les trémies. Le DHP a quelques similitudes avec le VRP, qui est connu pour être un problème NP-difficile. En outre, évidemment, le model-checking est bien connu pour être exposé à l'explosion combinatoire.

Influence de la capacité des trémies et évaluation du nombre de chemins à explorer

On peut remarquer dans la figure 5.11 que les courbes C2R1 et C2R2 sont supérieures respectivement aux courbes C1R1 et C1R2. Cet état de fait n'est pas anormal, car C2 indiquant une plus grande capacité de trémie, cela signifie que pour les instances correspondant à C2, plus souvent que dans le cas C1, HM aura le choix entre aller à l'emplacement des bennes ou récolter un autre rang.

Prenons un exemple simple : la capacité de la machine ne permet de récolter qu'un seul rang à la fois (on note $C1 \equiv 1rang$), et on a une parcelle formée de 4

rangs (R1, R2, R3, R4). La figure 5.12 présente le début du graphe d'états d'exploration. HM commence à l'emplacement des bennes et elle peut récolter l'un des 4 rangs, dans l'orientation directe ou inverse (8 possibilités). Dès que HM a récolté ce rang, elle doit aller à l'emplacement des bennes (B) pour vider ses trémies qui sont pleines. Il reste alors 3 rangs à récolter. Le même raisonnement peut être réitéré pour la récolte d'un nouveau rang. Au total, le graphe d'états sera composé de 1265 états : (1[l'emplacement des bennes] + 8[1^{ier} rang récolté] + 8[l'emplacement des bennes] + 8*6[2^{ième} rang récolté] + 8*6[l'emplacement des bennes] + 8*6*4 [3^{ième} rang récolté] +8*6*4[l'emplacement des bennes]+ 8*6*4*2 [4^{ième} rangs récolté] +8*6*4*2[l'emplacement des bennes]. Et il y aura 384 chemins dans le graphe (8*6*4*2), c'est à dire 384 façons de récolté le champ.

De façon générale, dans ce cas simple, si n est le nombre de rangs, alors le nombre exact d'états est $1 + \sum_{k=2}^{n} 2^{k+1}.k!$ [5] et le nombre de chemins est $2^n.n!$

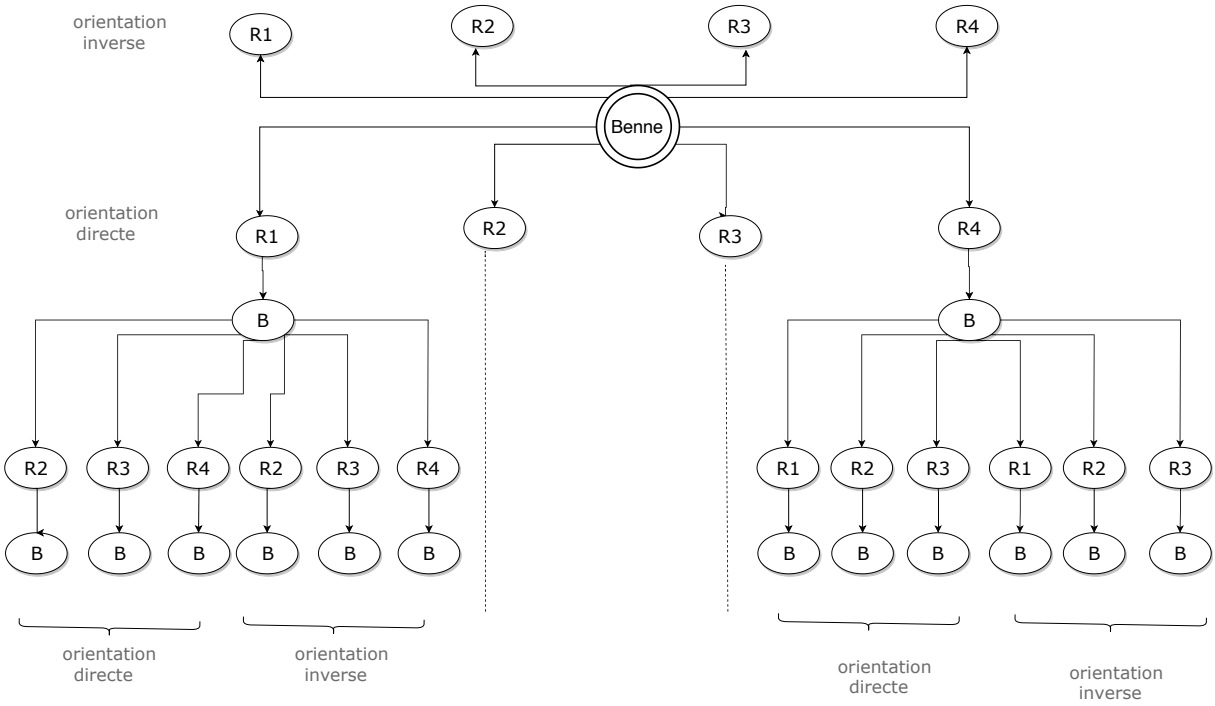


FIGURE 5.12 – Exemple de graphe d'exploration pour $n = 4$ et $C1 \equiv 1$ rang

Supposons maintenant que, comme dans la réalité, la capacité de la machine soit suffisamment grande pour récolter plusieurs rangs à la suite (ici on ne précise pas la limite, HM peut donc a chaque fois soit récolter un autre rang soit vider ses trémies). Comme dans l'exemple précédent, HM commence à l'emplacement des bennes et elle a 4 rangs à récolter dans l'orientation directe ou inverse (8 possibilités). Si HM

5. Le 1 initial s'explique car le point de départ de HM est imposé.

récolte un rang, elle a alors le choix entre aller à l'emplacement des bennes pour vider ou alors récolter un des 3 rangs non récoltés, dans le sens inverse du précédent (soit en tout 4 possibilités). De l'emplacement des bennes, il reste 3 rangs à récolter, avec la possibilité de choisir entre récolter selon l'orientation directe ou l'orientation inverse, soit 6 possibilités. En ayant récolté 2 rangs sur les 4, nous avons déjà identifié $8\times(3+6)$ chemins. En continuant, le même raisonnement peut être à nouveau réalisé à partir du choix qui aura été effectué. On arrive au final a 1296 chemins possibles $(8 \times (6 + 3) \times (4 + 2) \times (2 + 1))$.

De façon générale, dans ce cas simple, si n est le nombre de rangs, alors le nombre de chemins est $2^n \times \prod_{k=1}^{n-1}(2^k + k)$, ce qui est nettement supérieur au nombre de chemins pour $C1 \equiv 1rang$, et ce qui confirme les résultats de la figure 5.11 et de la table 5.2.

Influence du seuil de récolte de raisins de qualité A

Nous pouvons également observer sur la figure 5.11 que pour une même capacité de trémie, la courbe avec le code R1 est plus élevée que celle avec le code R2. Cela résulte du fait qu'une fois que la quantité *Rmin* de raisins A a été récoltée, HM passe à une récolte mixée, qui est plus complexe en nombre d'états. En effet, en mode mixé, les raisins A et B peuvent être mélangés dans les deux trémies, ce qui peut permettre à HM de traiter plus de rangs entre 2 vidanges. On se retrouve un peu dans le même cas que sur l'exemple simple illustré précédemment : en mode mixé, pour chaque rang, HM peut choisir soit de récolter un autre rang soit d'aller à la benne. Lorsque *Rmin* est plus petit, HM peut basculer plus tôt en mode mixé, donc le nombre de chemins est plus grand et par conséquence, le nombre d'états explorés est plus important.

Nous avons pu constater qu'avec l'outil dont nous disposons nous ne pouvons résoudre le problème de DHP que pour un nombre maximal de rangs égal à 9. Afin de pallier cette limitation nous allons maintenant faire appel à une fonctionnalité proposée par UppAal-CORA, la fonction de remaining, qui permet d'optimiser le nombre de chemins parcourus. Cette fonction est expliquée ci-après.

5.5.3 Modélisation avec ajout de la fonction de remaining

La performance de UppAal-CORA peut être améliorée en calculant pour chaque état une borne inférieure estimée du coût restant. Cette borne inférieure est mémorisée dans une variable réservée appelée *remaining*. Cette information permet à l'algorithme de model-checking de ne pas explorer jusqu'au bout une solution potentielle dont le coût s'annonce trop élevé pour représenter une solution optimale possible [15]. La fonction *remaining* permet donc de réduire le nombre d'états explorés de manière spectaculaire et par conséquence améliorer l'efficacité de la résolution. La fonction de remaining ne modifie pas la complexité globale maximale en termes

de nombre de chemins potentiellement explorables. Elle raccourcit seulement l'exploration en profondeur de l'arbre des chemins potentiels, par évaluation continue d'une borne inférieure du coût total d'un chemin incluant le début du chemin en cours d'évaluation.

5.5.3.1 Fonction de remaining

Pour le modèle DHP_PTA et à chaque état, la variable $remaining$ a été calculée en utilisant la formule suivante qui représente une borne inférieure estimée du coût que pourrait représenter la suite du chemin en cours.

$$\begin{aligned} remaining \;=\; & HarvestingTime \\ & + nEmptyingNonSelec \times DumpTime \\ & + nEmptyingNonSelec \times (2 \times minDBin) - factDBin \\ & + (nbRowNotHarvested - nEmptyingSelec) \times \; dmin + factdmin \end{aligned} \tag{5.2}$$

Sachant que :

- **HarvestingTime** : est le temps de récolte de la totalité des rangs non encore récoltés.
- **nbRowNotHarvested** : est le nombre de rangs non encore récoltés.
- **nEmptyingNonSelec** : est la limite inférieure du nombre de vidanges de trémies restant, calculé en fonction des capacités des trémies et des quantités des raisins non encore récoltées. En d'autres termes, nEmptyingNonSelec est calculé comme si la récolte s'effectuait en mode non sélectif et que les raisins A et B étaient mélangés dans une seule trémie de capacité 2 x $CmaxHopper$.
- **nEmptyingSelec** : est la limite inférieure du nombre de vidanges de trémies restant calculé comme si la récolte s'effectuait en mode sélectif. On a $nbRowNotHarvested \leq nEmptyingSelec \leq nEmptyingNonSelec$. La valeur de $nEmptyingSelec$ est toujours inférieure ou égale à $nbRowNotHarvested$ et supérieure ou égale à $nEmptyingNonSelec$.
- **DumpTime** : est le temps de vidange des deux trémies dans la benne.
- **minDBin** : est le temps de trajet minimum entre l'emplacement des bennes et un rang parmi ceux non encore récoltés.
- **factDBin** : est un facteur correctif du temps de trajet entre l'emplacement des bennes et un rang parmi ceux non encore récoltés. Il vaut 0 si HM est dans l'emplacement des bennes et il vaut $minDBin$ si HM est dans un rang.
- **dmin** : le temps de trajet minimum entre deux rangs non encore récoltés.

- **factdmin** : est un facteur correctif du temps de trajet entre deux rangs. Il vaut *dmin* si HM est dans un rang et il vaut 0 dans les autres cas.

Une illustration de ces variables est donnée dans la figure 5.13. Avant de commencer la récolte, le nombre de rangs non encore récoltés $nbRowNotHarvested$ est 4, le nombre minimum de vidange calculé en mode sélectif en fonction des capacités des trémies $nEmptyingSelec$ est 2 et le nombre minimum de vidange calculé en mode mixé en fonction des capacités des trémies $nEmptyingNonSelec$ est 2.

Il est aisé de vérifier que la fonction de *remaining* est une borne inférieure du coût restant, tous les termes constituant le calcul du remaining étant soit des minimums des valeurs, soit des valeurs exactes.

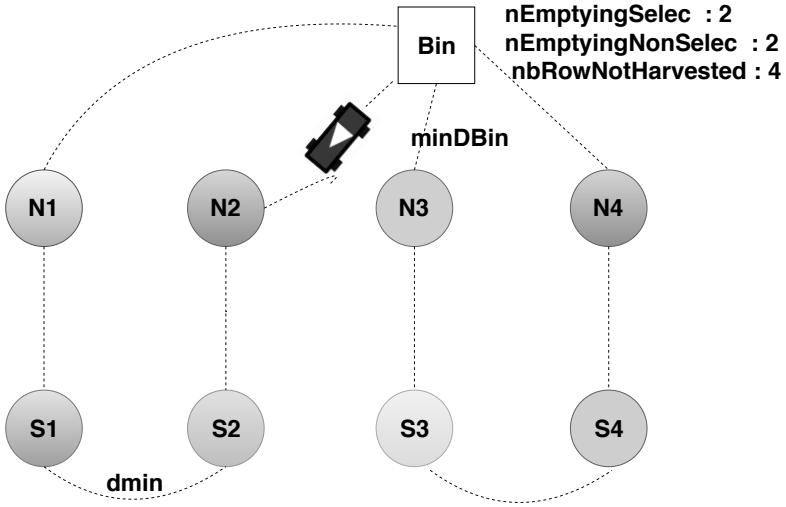


FIGURE 5.13 – Exemple illustratif de calcul de la fonction de *remaining*

5.5.3.2 Résultats du problème de DHP avec remaining

Le tableau 5.4 montre le résultat de la résolution du problème de DHP avec la gestion du coût restant, pour un ensemble d'instances de modèle défini par nRCiRi, avec $nCiRi$ est le même codage défini précédemment en section 5.5.2 [6]. On ajoute le R dans le nom de l'instance pour désigner l'utilisation de la fonction de Remaining. L'utilisation de *remaining* a permis de résoudre le problème DHP pour les 12 premiers rangs du champ étudié.

Les informations suivantes sont données pour chaque instance : nombre d'états explorés (*States*), coût total qui est le temps de récolte et de déplacement en secondes (*Cost*), temps de vérification de la requête dans UppAal-CORA en secondes (TE), taille maximale de mémoire nécessaire pour cette vérification en KBytes (*Max RSS*)

6. n est le nombre de rangs dans le champ ($n \in [7, 12]$), Ci la capacité maximale de la trémie en litres avec $C1 = 1000l$ et $C2 = 2000l$, Ri spécifie la valeur de $Rmin$ avec $R1 = 50\%$ et $R2 = 70\%$

et Facteur de Gain (FG) qui est le rapport du nombre d'états explorés sans ajout de la fonction de remaining au nombre d'états explorés avec ajout de la fonction de remaining (comparaison avec la table 5.2).

TABLE 5.4 – Résultats expérimentaux avec la gestion du coût restant

Instance	States	Cost	TE (s)	RSS (KB)	FG
7RC1R1	32019	655	0.31	7684	8
7RC1R2	33189	665	0.30	7076	6
7RC2R1	78035	655	0.51	11680	5
7RC2R2	**66045**	**655**	**0.43**	**10512**	5
8RC1R1	**8031**	**669**	**0.10**	**88**	117
8RC1R2	13552	676	0.21	6388	53
8RC2R1	50771	668	0.51	12408	40
8RC2R2	50487	668	0.51	11908	33
9RC1R1	974467	827	6.76	123252	9
9RC1R2	449599	827	3.48	60148	9
9RC2R1	2591458	809	17.45	305068	7
9RC2R2	1918631	809	13.30	220080	6
10RC1R1	181584	844	1.86	35916	–
10RC1R2	154027	844	1.63	30212	–
10RC2R1	800879	825	8.63	147708	–
10RC2R2	793346	825	8.43	144460	–
11RC1R1	29861147	990	203.31	3818508	–
11RC1R2	7659725	990	69.33	994700	–
11RC2R1	29557772	967	255.38	4105804	–
11RC2R2	30626152	967	266.12	4055288	–
12RC1R1	1598213	1007	19.18	297832	–
12RC1R2	1639723	1007	20.11	300416	–
12RC2R1	12823061	983	157.40	2429152	–
12RC2R2	12583816	983	156.28	2370280	–

A) Gain en performance

Comparé aux résultats obtenus sans considérer le coût restant, le nombre d'états explorés a été fortement réduit, avec un facteur de 5 pour 7C2R2 et jusqu'à un facteur de 117 pour 8C1R1. Ces chiffres confirment donc le gain en complexité que nous espérions. Ce gain dépend évidemment et fortement des paramètres du problème (seuil de récolte de raisins de qualité A, capacité des trémies). Il dépend également de la précision et de l'efficacité de la fonction de remaining, chacun des chemins coupé étant une source de complexité. Dans la suite, une analyse de l'influence de la précision de la fonction de remaning est fournie.

B) Influence de la précision de la fonction de remaining
Dans le tableau 5.4, on peut observer que, pour une même configuration de paramètres, le nombre d'états explorés est plus élevé lorsque le nombre de rangs n est impair que lorsque le nombre de rangs est pair. Cela résulte de la précision de la borne inférieure offerte par notre fonction de remaining. En effet, l'équation de *remaining* est en partie basée sur le trajet le plus court d'une extrémité d'un rang à l'emplacement des bennes ($minDBin$). Généralement, ce trajet est égal à la distance (ou le temps de parcours) entre l'emplacement des bennes et l'extrémité du rang non récolté la plus proche de l'emplacement des bennes. Dans notre cas les extrémités *Nord* sont les plus proches de l'emplacement des bennes. Dans le cas d'un nombre impair de rangs, et comme l'illustre la figure 5.14, il y a un trajet à effectuer entre l'emplacement des bennes et une extrémité *Sud* d'un rang non récolté (trajet en rouge entre Bin et $S3$ sur la figure 5.14). Ce temps de déplacement est beaucoup plus long que $minDBin$, et la borne inférieure choisie pour la fonction de *remaining* est alors plus éloignée du coût restant minimum réel. Cette sous estimation peut conduire à ne pas rejeter assez tôt les chemins peu prometteurs, et donc à explorer plus d'états.

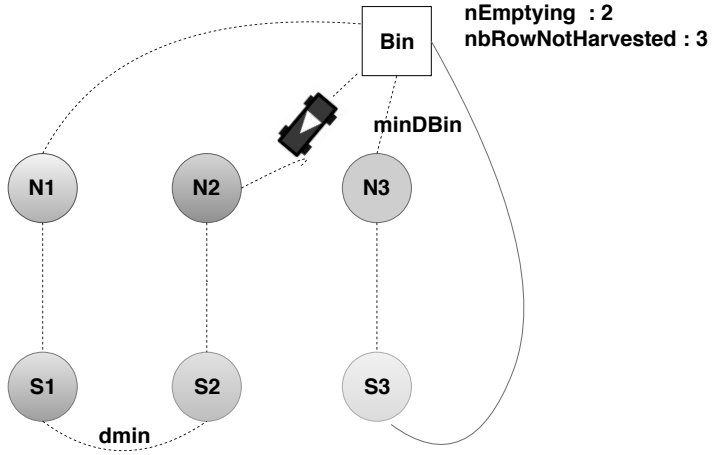


FIGURE 5.14 – Exemple d'un nombre de rang impair

Les instances avec une capacité de trémie élevée sont plus difficiles à résoudre que les instances avec une capacité de trémie inférieure. En effet, malgré l'ajout d'une fonction remaining, les phénomènes décrits à la section 5.5.2.2 sont encore applicables.

Plus la fonction de remaining est proche de la valeur exacte du coût restant, plus elle sera efficace. Il n'est toutefois pas toujours possible de la calculer de façon exacte. De plus, il faut trouver le juste équilibre entre la complexité de calcul de cette fonction qui est effectuée dans tous les états du graphe, et la complexité globale du processus de validation.

5.5.4 Résolution du DHP : Comparaison du model checking avec une approche de l'état de l'art [37]

L'approche décrite par N. Briot dans ses travaux ([35, 36, 37, 38]), et déjà évoquée plus haut dans ce manuscrit, a les mêmes objectifs que nous de résolution du problème DHP, mais utilise des techniques de programmation par contraintes. Nous désirons pousser un peu la comparaison entre nos deux approches. Dans cette partie, nous rappelons quelques résultats rapportées dans [37]. Il faut noter que les besoins en mémoire nécessaire pour la résolution des instances n'ont pas été indiqués dans [37] donc nous nous limiterons à une comparaison en temps d'exécution.

Nous commençons avec les résultats donnés dans le tableau 1 de [37], que nous reportons ici dans la table 5.5. Ce tableau indique le temps moyen nécessaire pour résoudre les instances comportant 10 et 12 rangs avec le modèle *Step* et le modèle *Successeur* implémentés avec le solveur Choco. Nous avons effectué des expériences pour résoudre les mêmes instances avec notre modèle DHP_PTA : nous avons utilisé les mêmes valeurs pour les capacités des trémies ($1000l$ et $2000l$) et les mêmes valeurs de Rmin (50% et 70% de raisins de qualité supérieure). Dans la table 5.5, nous avons ajouté nos résultats à ceux de Briot et al. Le résultat fourni pour chaque nombre de rangs est la moyenne des 4 instances C1R1, C1R2, C2R1 et C2R2.

TABLE 5.5 – Comparaison du modèle *Step*, du modèle *Successeur* et du modèle DHP_PTA pour 10 et 12 rangs (moyenne des instances C1R1, C1R2, C2R1, C2R2)

Nombre des rangs	*Step*	*Successeur*	DHP_PTA
10	180 s	8 s	2.22 s
12	2749 s	118 s	49.25 s

D'après la table 5.5, il est clair que le modèle DHP_PTA présente de bonnes performances par rapport au modèle *Step* et *Successeur*.

Nous poursuivons avec les résultats donnés dans le tableau 2 de [37]. Ce tableau indique le temps nécessaire pour résoudre les instances comportant 12 rangs avec le modèle *Successeur* implémentés avec le solveur Choco. Les paramètres considérés pour les capacités des trémies et le seuil Rmin sont les mêmes que précédemment. Nous avons effectué des analyses pour résoudre les mêmes instances avec notre modèle DHP_PTA. Dans la table 5.6, nous ajoutons nos résultats à ceux de Briot et al. Cette table montre qu'avec l'approche de N. Briot avec le modèle *Successeur*, les instantes avec une grande capacité de trémie sont plus faciles à résoudre que celles avec une petite capacité de trémie. Ce comportement est inverse à celui observé avec notre méthode.

Nous continuons avec les résultats donnés dans le tableau 4 de [37], reportés dans la table 5.7. Ce tableau compare deux approches développées par N Briot : la première est le modèle *successeur* et la deuxième se base sur la programmation

TABLE 5.6 – Résultats expérimentaux des deux approches pour 12 rangs

Approche	Instance	TE (s)
N Biot et al. (modèle successeur)	12C1R1	227
	12C1R2	170
	12C2R1	35
	12C2R2	41
R. Saddem et al.	12RC1R1	19.18
	12RC1R2	20.11
	12RC2R1	157.40
	12RC2R2	156.28

linéaire en nombre entier, avec un modèle appelé ILP (Integer linear problem). Les deux modèles sont implémentés avec le solveur Cplex. Le tableau 4 indique le temps nécessaire pour résoudre, avec les deux modèles, l'instance comportant 12 rangs avec les paramètres $1000l$ pour la capacité des trémies et 70% pour le seuil Rmin et il indique également le temps de récolte obtenu par cette instance. Nous avons effectué des analyses pour résoudre la même instance avec notre modèle DHP_PTA. Dans la table 5.7, nous ajoutons nos résultats à ceux de Briot et al.

TABLE 5.7 – Résultats expérimentaux des deux approches (N. Briot et al. et R. Saddem et al.) pour l'instance 12 rangs

Approche	Instance	TE (s)	temps de Récolte
N Biot et al. (modèle successeur)	12C1R2	496	960
N Biot et al. (modèle ILP)	12C1R2	3	960
R. Saddem et al.	12RC1R2	20.11	1007

D'après la table 5.7, pour l'instance 12RC1R2, la requête UppAal-CORA pour le DHP a été résolue en 20.11 s sur un processeur Intel (R) Xeon (R) CPU E5-2667 3.20 GHz. La résolution de la même instance, sur un processeur Intel (R) Xeon (R) E5-2697 cadencé à 2,60 GHz, avec le modèle *successeur* a pris 496 s et elle a pris 3 s avec le modèle ILP, ce qui est nettement meilleur. Cependant, d'après [37], le modèle ILP est efficace seulement pour les instances comprenant un petit nombre de rangs. Lorsque le nombre de rangs augmente, le modèle ILP ne trouve pas de résultats.

Nous avons également essayé de comparer le temps de récolte optimal obtenu, sans toutefois pouvoir le faire précisément. En effet, par exemple pour résoudre l'instance 12RC1R2, notre approche trouve que le temps de récolte optimal est égal à 1007s alors que N. Briot annonce la valeur 960s dans [37] et 1360s dans [35]. Or nous n'avons pas réussi à obtenir suffisamment d'informations détaillées pour

être en mesure de comparer précisément le temps de récolte optimal obtenu par ces deux approches. Il est cependant important de noter que les résultats de nos deux approches évoluent de façon cohérente et relativement similaire sur les instances avec un nombre de rang pair (il n'y a pas de résultats disponibles avec leur méthode pour des instances comprenant un nombre impair de rangs).

En conclusion, le modèle DHP_PTA offre de bonnes performances. Avec notre modèle, nous sommes capables de résoudre des instances comprenant 12 rangs. Au delà, un problème d'explosion combinatoire se produit. Les modèles *step* et *successeur* permettent de résoudre également des instances comprenant 12 rangs. Au delà, le modèle *successeur* trouve des solutions sous-optimales. Pour certaines instances, notre approche est mieux que celles de N. Briot et pour d'autres, elle est moins bonne [7].

Outre la performance, une différence importante dans la représentation du problème entre l'approche de N. Briot et co-auteurs et la notre, est que dans la première, des sommets temporels imaginaires représentant le dépôt pour chaque vidange sont ajoutés au graphe. Le nombre de ces sommets imaginaires est égal au nombre de vidanges, qui est fixé a priori. Dans notre cas, le nombre de vidange n'est pas fixé à l'avance, car il peut influencer les résultats et doit donc rester une variable du processus. De plus, l'intérêt de notre approche est que le même modèle permet potentiellement la vérification de plusieurs propriétés (voir section 5.4.4). Le potentiel va donc au delà de la vérification (et optimisation) du temps de récolte. Notre approche est une approche générique, elle a été appliquée dans le chapitre précédent pour vérifier et optimiser la pulvérisation. Elle pourrait être appliquée pour traiter plusieurs problématiques différentes issues de l'Agriculture de Précision.

5.5.5 Passage à l'échelle

Nous avons vu que l'approche de résolution par model checking du problème de DHP est intéressante mais qu'elle reste encore limitée par rapport aux problèmes de taille réelle. Nous examinons donc ici l'applicabilité de notre approche pour générer des logistiques de récolte sur des parcelles viticoles typiques. Il n'est en effet pas rare d'avoir des parcelles comportant plus d'une vingtaine de rangs.

La première limitation est purement technique et concerne le fichier binaire exécutable actuellement disponible pour UppAal-CORA. Celui-ci s'appuie sur une architecture 32 bits, limitant ainsi l'utilisation de la mémoire à 4 Go de RAM. Avec un binaire compilé pour une architecture 64 bits, et un ordinateur bien pourvu en mémoire, il aurait été possible de résoudre, avec la méthode présentée, des instances comportant plus de 12 rangs.

7. Nous rappelons que les besoins en mémoire nécessaire pour la résolution des instances dans travaux de N. Briot & co n'ont pas été indiqués.

La seconde limitation touche au modèle présenté dans ce chapitre, qui, par choix, a été développé pour être le plus proche possible du modèle *Step* proposé par Nicolas Briot, afin d'établir une comparaison qualitative équitable des performances. Notre modèle pourrait être légèrement optimisé en mémoire, par exemple en supprimant certaines variables. Ce type d'optimisation, bien que pouvant avoir un impact intéressant, ne changera cependant pas la problématique de la complexité exponentielle du nombre de chemin et d'états à explorer.

Mais la limitation principale est l'explosion combinatoire intrinsèque à la manipulation d'un outil de model-checking comme UppAal-CORA, dont le principe réside dans un parcours potentiellement exhaustif des chemins autorisés par les contraintes sur la dynamique du système. Le principe de recherche d'un chemin optimum, et la fonction de *remaining* contribuent à réduire cette exploration, sans toutefois garantir qu'un problème de grande taille puisse être traité.

5.6 Conclusion

Dans ce chapitre, nous avons étudié comment résoudre le problème de vendange sélective (DHP - Differential Harvest Problem), en utilisant la fonctionnalité "Cost Optimal Reachability Analysis" implémentée dans l'outil de Model-checking UppAal-CORA. Nous avons conçu un modèle de DHP, basé sur le formalisme des automates temporisés de coût et l'outil UppAal-CORA. Tous les détails du modèle et de la propriété qui permet de résoudre le DHP ont été expliqués.

Nous avons dans un premier temps illustré notre méthodologie sur une parcelle de vigne réelle, vignoble située à l'INRA de Pech-Rouge. Ces premiers résultats ont montré que notre méthodologie fournissait bien une borne optimale du temps de récolte, mais que la complexité intrinsèque du model checking ne permettait pas de répondre au problème en considérant la totalité des rangs de la parcelle.

Nous avons ensuite utilisé la fonctionnalité d'optimisation offerte par UppAal-CORA qui permet de réduire le nombre de chemins parcourus en considérant une valeur minimale du coût restant (la fonction de *remaining*). L'importance pratique de l'utilisation d'une limite inférieure suffisamment proche du coût restant réel a été soulignée par nos expériences. L'utilisation de cette fonction de façon efficace n'était pas simple et nécessite une grande expertise, une forte intuition, une compréhension approfondie du système et de la rigueur scientifique pour définir une borne inférieure. La précision de son estimation permet un filtrage plus ou moins efficace des états de l'arbre influençant fortement la performance de l'algorithme de résolution.

En premier lieu, notre approche a été comparée à une approche classique qui consiste à répéter un motif régulier jusqu'à le parcours et la récolte de toute la parcelle. Les résultats montrent que notre approche permet d'économiser jusqu'à 9% de la durée totale du parcours. En deuxième lieu, notre approche a été compa-

rée à une approche de l'état de l'art traitant du même problème, en utilisant des techniques de programmation par contraintes. Dans certains cas, notre approche améliore significativement le temps d'exécution.

Cependant, notre approche ne permet pas de résoudre des instances comportant plus de 12 rangs pour des raisons liées : D'abord, aux limites de l'outil de vérification, ensuite, à la nature du problème traité qui génère un nombre de chemin exponentiel et enfin au MC qui explore exhaustivement l'espace d'état pour générer le parcours optimum.

Dans le chapitre suivant, nous allons continuer à explorer les possibilités offertes par le model-checking pour l'agriculture de précision en étudiant le parcours d'un robot autonome dans un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques.

6 Environnement ouvert : Actions spatiales et model checking

Sommaire

6.1 Introduction

Les deux chapitres précédents présentent des méthodes de vérification et d'optimisation pour deux exemples d'application concrets en agriculture de précision. Le problème traité dans le chapitre 4 permet de choisir la commande pour la pulvérisation de précision dans un rang de vigne, ce qui correspond à une représentation spatiale que nous avions qualifié de '1D', tandis que le problème traité chapitre 5 permet d'optimiser le temps de récolte d'une parcelle de vigne, ce qui correspond à une représentation spatiale que nous avions qualifié de '1D+'. Dans ces deux problèmes, l'engin ou le robot mobile se déplace dans un environnement structuré par des rangs.

Nous abordons dans ce nouveau chapitre la problématique d'un environnement spatial du robot peu ou pas structuré. Nous allons ici considérer les déplacements d'un robot mobile dans un environnement ouvert, c'est-à-dire en 2D.

Le déplacement autonome d'un robot agricole en milieu ouvert n'a pas encore atteint un niveau de maturité opérationnel en grande partie parce que le niveau de réactivité et d'intelligence des robots agricoles est encore limité [1]. Aujourd'hui, grâce au développement des capteurs (Lidar, GPS...), un robot agricole est capable de suivre une trajectoire dans un environnement structuré, ou de suivre un piéton "leader". Par exemple, l'objectif principal du projet ANR Astrid Baudet-Rob [2] a été la conception d'un robot capable de suivre en complète autonomie une ou un groupe de personnes, tout en transportant une partie de leur matériel.

Pour lever la question de l'autonomie, nous allons supposer dans ce chapitre que le robot évolue dans un environnement connu. Les actions que le robot peut effectuer au cours de sa mission, qu'elles aient un effet sur l'environnement ou non, dépendent de sa position, de son état interne et de son environnement.

Nous supposons que l'environnement ouvert peut être représenté comme une grille 2D. Comme indiqué au chapitre 3, le robot est supposé avoir son stationnement initial habituel à un endroit précis dans l'exploitation. Il lui faut alors atteindre un champ pour effectuer une opération, et le cas échéant passer d'un champ à un autre. Considérons ce dernier cas. ll y aura beaucoup de points possibles pour le départ dans le premier champ et beaucoup de points possibles pour l'arrivée dans le deuxième champ. On souhaite que le changement entre les champs soit robuste, c'est-à-dire qu'il ne dépende pas d'un point unique de départ du premier champ et d'un point unique d'arrivée dans le deuxième. Dans l'espace qui sépare les deux champs, modélisé comme une grille 2D, le chemin du robot mobile sur cette grille est donc considéré comme inconnu a priori. Notre objectif est de proposer une méthode qui permette de vérifier que l'atteignabilité du champ en vue de débuter une opération agricole est garantie. Afin d'incorporer une question de lien entre l'état interne du robot et des actions spatialisées dans le problème traité, qui reste un problème abstrait pour développer une recherche méthodologique sur un environnement 2D, nous supposons que le robot doit effectuer une opération unique sur chaque case de la grille sur laquelle il se déplace, et que des contraintes de précédence s'exercent entre types d'opération.

Le chapitre est organisé comme suit. Le problème d'étude est défini de façon précise dans la section suivante. Puis, dans la section 6.3, une modélisation du problème d'étude sous la forme d'un réseau d'automates temporisés est décrit. La propriété d'accessibilité étudiée est ensuite définie. Dans la section 6.4, le principe de vérification de la propriété d'accessibilité est présenté. Les résultats expérimentaux sont décrits et discutés en section 6.5. Une conclusion clôt le chapitre.

1. `http://www.irstea.fr/la-recherche/unites-de-recherche/tscf/robotique-mobilite-environnement-agriculture`
2. `http://actions-territoires.irstea.fr/sol/robotique-agricole-baudet-rob`

6.2 Définition du problème d'étude

6.2.1 Cadre général

Le problème étudié dans cette section s'énonce comme suit :
Un robot se déplace sur une grille régulière en respectant des contraintes de mouvement. Dans chaque case de la grille, le robot effectue une action parmi l'ensemble de celles autorisées pour cette case. Ces actions sont soumises à des contraintes de précédence, afin de représenter les contraintes de dynamique interne du robot. Chaque case de la grille ne peut être visitée qu'une seule fois (économie de visites).

6.2.2 Définition détaillée de l'exemple d'étude

On considère un robot qui se déplace sur une grille régulière rectangulaire de taille $longueur \times largeur$. Le robot part du bord gauche de la grille. Les contraintes de mouvement sont :

- les déplacements vers la droite, le haut, et le bas, sont autorisés (sauf pour les cases en bord de grille) ;
- les déplacements vers la gauche, qui représentent un retour arrière dans la grille, sont interdits.

Chaque case de la grille est représentée par un couple (x,y) avec x comme indice de ligne et y comme indice de colonne. Chaque case ne peut être visitée qu'une seule fois. La figure 6.1 décrit la représentation d'une grille de taille 32 (longueur = 8 et largeur =4). Par souci de simplification, nous considérerons à partir d'ici qu'une seule action possible est spécifiée pour chaque case. Un exemple de répartition des actions sur la grille de taille 32 est décrit dans la figure 6.2.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8

FIGURE 6.1 – Représentation de la grille, avec l'indice des cases.

A	B	B	A	B	C	A	C
C	C	A	A	B	C	C	A
C	C	A	A	C	A	B	A
C	C	A	A	C	B	B	A

FIGURE 6.2 – Répartition des actions dans la grille.

Soit A, B et C les actions à effectuer par le robot dans le champ. Ces actions sont soumises à des contraintes de précédence. X $\longrightarrow$ Y signifie "effectuer l'action Y après avoir effectué l'action X". Dans notre cas, les contraintes de précédence pour les actions A, B et C sont décrites de la manière suivante :

- A $\longrightarrow$ A : autorisé
- A $\longrightarrow$ B : autorisé
- A $\longrightarrow$ C : non autorisé
- B $\longrightarrow$ A : non autorisé
- B $\longrightarrow$ B : non autorisé
- B $\longrightarrow$ C : autorisé
- C $\longrightarrow$ A : autorisé
- C $\longrightarrow$ B : non autorisé
- C $\longrightarrow$ C : autorisé.

Ces actions de précédence peuvent être représentées par un automate (figure 6.3). On remarque que les actions A et C sont des actions stables [3]. L'action B est une action de transition, elle ne peut être répétée deux fois de suite.

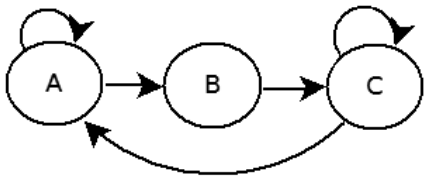


FIGURE 6.3 – Automate des contraintes de précédence de notre exemple

6.2.3 Exemple de fonctionnement du robot

Supposons que initialement le robot se trouve dans une des cases du bord de gauche (encadré en vert dans la figure 6.1) et qu'il doive se déplacer jusqu'au bord de droite (encadré en bleu dans la figure 6.1). Supposons qu'il soit initialement dans la case (1,1). Dans sa case de départ, le robot doit réaliser l'action A (figure 6.2). Ensuite, les deux déplacements possibles sont vers le bas (l'action de la case (2,1) est C) ou vers la droite (l'action de la case (1,2) est B). Les deux cases (1,2) et (2,1) n'étant pas encore visitées par le robot, les contraintes d'économie de visite sont vérifiées. Il reste à s'assurer que les contraintes de précédence le sont aussi. Or faire l'action C précédée de la réalisation de l'action A viole les contraintes de précédence donc le robot n'est pas autorisé à se déplacer vers le bas. Par contre, l'action B peut succéder à l'action A. La contrainte de précédence étant vérifiée le robot peut se déplacer vers la droite dans la case (1,2). Puis le robot ne pourra se déplacer que vers le bas, en case (2,2), pour effectuer l'action C (la succession des actions B ->B est interdite, et il n'a pas le droit de revenir sur la gauche). Et ainsi de suite.

3. Une action stable est une action qui ne change pas nécessairement après un mouvement.

6.3 Modélisation du problème et propriété à vérifier

6.3.1 Modélisation du problème

Pour étudier le problème d'un robot agissant localement selon une grille spatiale, nous avons mené une étude et proposé plusieurs modèles. Le but de cette étude est de chercher si la manière de modéliser avait une influence sur la vérification du comportement du robot. Nous avons retenu le Modèle Région de Déplacement (MRD) qui est simple à mettre en œuvre et permet aisément de faire varier la taille de la grille. Ce modèle représente par des états les directions de déplacement, et la position du robot par des entiers.

La position actuelle du robot dans le modèle est notée (i, j). Elle est initialisée à (i_{init}, j_{init}). Le modèle MRD est composé de deux automates : un automate de commande et un automate de déplacement. Dans la suite, le modèle MRD est décrit en détails.

6.3.1.1 Définition des constantes, des variables et des fonctions

A) Définition des constantes

- $length$: est la longueur de la grille (en nombre de colonnes).
- $width$: est la largeur de la grille (en nombre de lignes).
- $NbrAction$: est le nombre d'actions que le robot peut effectuer.
- $Action[width][length]$: est un tableau de type entier de taille largeur $\times$ longueur. $Action[i][j]$ représente l'action que le robot doit effectuer dans la case (i, j) de la grille lorsqu'il la visite. L'action A est représentée par 1, l'action B par 2 et l'action C par 3.

B) Définition des variables

- (i, j) : représente la position courante du robot. i pour la ligne et j la colonne.
- (i_init, j_init) : représente la position de la case de départ du robot.
- (i_fin, j_fin) : représente la position de la case finale du robot.
- $visit[width][length]$: est un tableau de booléens de taille largeur $\times$ longueur, qui représente la mémoire de visite des cases de la grille par le robot. Si une case (i, j) est visitée par le robot alors $visit[i-1][j-1]$ est mis à `vraie`. Au début, ce tableau est initialisé à `faux`.

- *depAut*[] : est un tableau de 3 booléens, qui représente les autorisations de déplacement vers les 3 directions potentielles. Cette variable permet d'éviter que l'automate de commande cycle indéfiniment dans une direction. *depAut*[0], et respectivement *depAut*[1] et *depAut*[2] gèrent l'autorisation de déplacement vers la droite, le haut et le bas. Au début, le robot peut se déplacer vers toutes les directions potentielles ($depAut[0] = depAut[1] = depAut[2] = true$). Supposons que le robot soit sur la frontière droite et qu'il veuille se déplacer vers la droite. L'automate de commande vérifie si à partir de la position courante et en fonction des contraintes de mouvement, le robot peut se déplacer vers la droite. Si ce n'est pas le cas alors *depAut*[0] est mis à jour à $false$ et cette direction ne sera pas testée une deuxième fois.

- *authorization* : est une variable booléenne. Elle est initialisée à `faux` et elle est mise à jour dans l'automate de commande. Cette variable sert à vérifier les contraintes de précédence.

- *bas*, *haut* et *droite* : ce sont des variables de type *chan*, c'est-à-dire des canaux de synchronisation qui synchronisent les différents automates lors des déplacements du robot vers le bas, le haut ou la droite.

C) Définition des fonctions :

- *void update*() : cette fonction est appelée par l'automate de déplacement après avoir effectué un déplacement vers une direction donnée. Elle ré-initialise le tableau *depAut* à la valeur `vraie` pour autoriser le robot à se déplacer vers n'importe quelle direction.

- *bool authorize*(*int A*, *int B*) : cette fonction vérifie les contraintes de précédence entre les actions A et B. Si les contraintes de précédence sont vérifiées elle retourne `vraie` sinon `faux`. La fonction *authorize* remplace l'automate de précédence décrit dans la figure 6.3.

6.3.1.2 Description de l'automate de déplacement

Cet automate (voir figure 6.4) représente le déplacement réel du robot et permet la mémorisation de ses déplacements. Il a 4 états : un état *Current* et 3 états qui désignent les déplacements (*Haut*, *Bas* et *Droite*). L'état initial *Current* est un état stable qui représente l'état du robot entre deux déplacements. C'est également dans cet état que s'écoule le temps, les autres états étant étiquetés par « committed », ce qui signifie qu'ils sont instantanés. L'automate de déplacement reçoit l'ordre de l'automate de commande, via les canaux de synchronisation *haut*, *bas* ou *droite*, pour se déplacer dans une direction déterminée. Après le déplacement, l'automate met à jour toutes les variables concernées : i et j pour refléter la nouvelle position du robot, le tableau *visit* pour marquer la case comme visitée, et la fonction *update* est appelée.

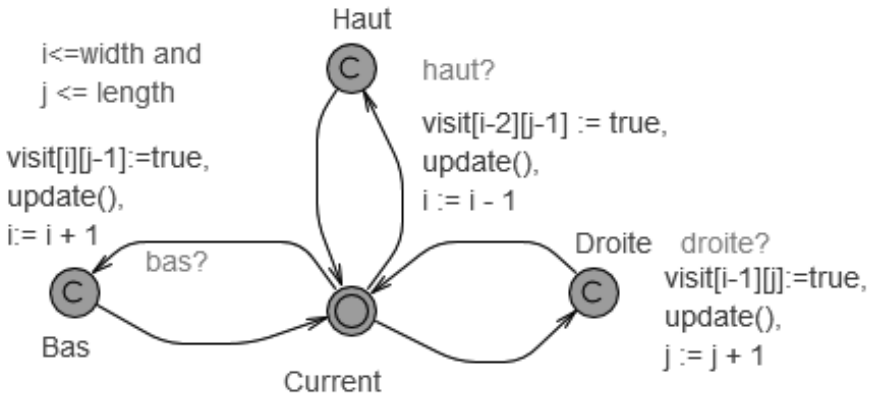


FIGURE 6.4 – Automate de déplacement

6.3.1.3 Description de l'automate de commande

L'automate de commande (voir figure 6.5) représente le contrôleur du robot. A l'état *Init*, l'automate choisit la case de départ i_init, j_init qui est obligatoirement un point du bord de gauche de la grille : j_init est fixée à 1 (1ère colonne) et la commande `p:int[1,width]` permet de choisir aléatoirement une valeur de ligne pour i_init (via la variable p). Après avoir choisi la case de départ, l'automate de commande cherche dans quelle direction il peut se déplacer. Il y a trois parties similaires qui correspondent à chaque direction potentielle. Pour simplifier, nous ne donnons les détails que du déplacement vers la droite. Dans l'état "Position", si $depAut[0]$ vaut `vrai` le contrôleur vérifie si, à partir de la position actuelle, le robot peut se déplacer vers la droite (i.e. il n'est pas en bout de grille). Si c'est le cas, l'état de contrôle peut changer en *"SeekNeighbor"*. Le contrôleur vérifie alors les contraintes de visite, i.e. si le voisin cible a déjà été visité, ce qui mène à l'état *"CheckAuthorization"*, puis les contraintes de précédence entre les actions des deux cases concernées par le déplacement. Si le déplacement est autorisé, l'automate retourne dans l'état *"Position"* et un signal de commande *droite*! est envoyé à *l'automate de déplacement*. Dans tous les autres cas, le chemin mène à l'état *"RightNotPermitted"*, et le contrôleur met à jour la variable *depAut* pour mémoriser que ce déplacement n'est pas possible et éviter de boucler indéfiniment sur ce test. De l'état *"RightNotPermitted"*, le contrôleur retourne directement à l'état *"Position"*.

Maintenant que le modèle MRD a été décrit en détails, la section suivante décrit les propriétés à vérifier sur ce modèle.

6.3.2 Propriété étudiée

Avec UppAal, de nombreuses propriétés peuvent être vérifiées sur notre modèle pour s'assurer de sa cohérence et de son comportement, par exemple :

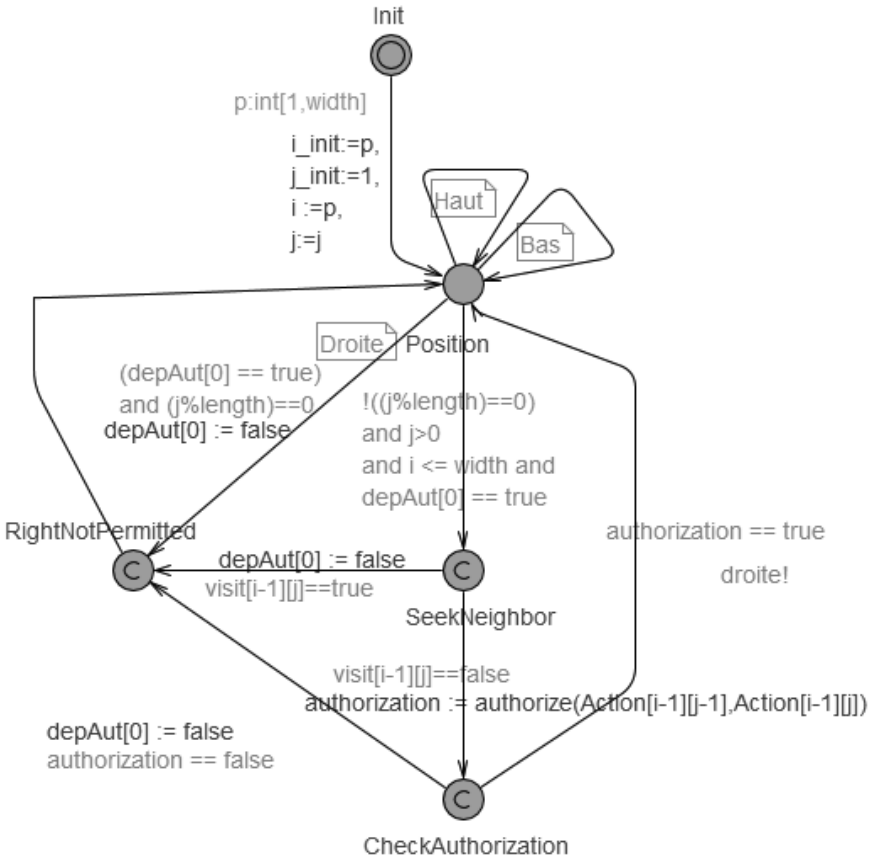


FIGURE 6.5 – Automate de commande

1. Est-ce-qu'il existe un blocage (*deadlock*) dans le modèle ?
   $PO1 : E <>\ deadlock$.
   La réponse est non.

2. Est-ce-qu'il existe au moins un chemin possible du bord de gauche au bord de droite ?
   $PO2 : E <> j == length$
   La réponse dépend de la distribution des actions dans la grille.

3. Est-ce-que le robot se trouve toujours dans une case de la grille ?
   $PO3 : A[]\ i <= width\ and\ j <= length$
   La réponse est oui.

Les propriétés précédentes sont des propriétés simples. Dans ce chapitre nous allons nous intéresser à une propriété plus complexe, que nous qualifierons de "propriété d'atteignabilité totale bord à bord d'une grille G" : *Sachant que B_G représente l'ensemble des cases du bord gauche de la grille G et B_D l'ensemble des cases du bord*

droit, en partant de n'importe quelle case de B_G, existe-t-il des chemins permettant d'atteindre toutes les cases de B_D ?

Cette propriété, que nous appellerons (Π), peut être exprimée formellement de la façon suivante [4] :

$$\forall\ g \in\ \mathcal{B}_{\mathcal{G}}(G), \forall\ d \in\ \mathcal{B}_{\mathcal{D}}(G), \exists\ g \longrightarrow d\ \ (\Pi)$$

Dans la suite, nous vérifions cette propriété sur le modèle MRD.

6.4 Résolution directe du problème de vérification de la propriété d'accessibilité

Dans cette section, nous présentons le principe de résolution directe du problème de vérification de la propriété d'accessibilité totale bord à bord. Cela consiste à résoudre la propriété étudiée sans aucune heuristique ou décomposition. Nous faisons appel à l'algorithme de référence présenté ci-après. Il est basé sur le fait que la propriété d'atteignabilité totale bord à bord Π peut être vue comme n^2 propriétés d'atteignabilité simples (avec n la largeur de la grille).

Algorithm 2 Algorithme Référence

```
1: function RÉFÉRENCE()
2:     Initialisation_Référence(B_G, B_D, GridB_DB_G);
3:     for e_1 in B_G do
4:         for e_2 in B_D do
5:             ch = CallUppaalRef(e_1, e_2)
6:             if ch == "OK" then
7:                 GridB_DB_G[e_1, e_2] = 1
8:             end if
9:         end for
10:    end for
11:    Analyse();
12: end function
```

Au début, l'algorithme *Référence* initialise les structures de données (B_G, B_D, $GridB_DB_G$) : B_G et B_D sont des listes qui contiennent respectivement les points du bord de gauche et de droite de la grille. $GridB_DB_G$ est une matrice de taille fixe dont tous les éléments sont initialisés à -1. Elle servira à mémoriser l'existence des trajets en partant du bord de gauche pour arriver au bord de droite.

4. Par souci de clarté, nous ne donnerons pas son expression dans la logique TCTL utilisée par UppAal.

Ensuite, pour chaque point e_1 du bord gauche B_G et chaque point e_2 du bord droit B_D, la fonction *CallUppaalRef* est appelée. Cette fonction a la signature suivante *CallUppaalRef(source, destination)*. Elle prend comme paramètres la source (point du bord gauche) et la destination (point du bord droit) et elle retourne une chaîne de caractères *ch*. Elle permet de vérifier la requête suivante sur le modèle MRD :

`E<> i_init = i(`e_1`) and j_init = 1 and i_fin = i(`e_2`) and j_fin = lenght`

i(e_1) et i(e_2) signifie respectivement l'indice i du point e_1 et l'indice i du point e_2 et la requête cherche s'il existe un chemin du point e_1(i(e_1),1) au point e_2(i(e_2),lenght). La chaine *ch* peut avoir 3 valeurs possibles : "Ok" (si la requête est vérifiée), "NotOk" (si la requête n'est pas vérifiée) et "ExpCom" (si la requête ne peut être vérifiée à cause de l'explosion combinatoire). Si ch == "Ok" alors $GridRBLB[e_1, e_2]$ est mis à jour à 1.

La fonction *Analyse()* permet de vérifier si le problème global est résolvable ou non. En d'autres termes, elle permet de vérifier si la propriété d'atteignabilité totale bord à bord est satisfaite sur le modèle MRD ou pas. En effet, si $GridB_DB_G$ est remplie avec 1 pour tous les couples de points bord gauche/bord droit alors la propriété est vérifiée pour G.

6.5 Résultats et discussion

Dans cette section, nous étudions les performances de l'algorithme de référence. Nous nous concentrons sur son impact en besoins mémoire et en temps d'exécution. Tout d'abord, nous allons donner une estimation de la complexité de l'algorithme de référence. De façon générale, le robot peut avoir au plus n mouvements possibles (dans notre cas $n = 3$) depuis chaque case de la grille. La complexité de l'algorithme de référence par rapport à la taille de la grille w (largeur) $\times$ l (longueur) est donc n^{wl}. Sur une instance particulière du problème, la complexité réelle dépend du nombre de déplacements réellement possibles en fonction des contraintes d'actions et peut donc être inférieure. Par ailleurs, le nombre d'états à parcourir dans le système d'automates est en réalité supérieur au nombre de déplacements possibles car l'exécution des automates peut mener à plusieurs états pour la gestion d'un seul mouvement. Par exemple, sur l'automate de déplacement (figure 6.4), le changement de case du robot dans la grille est codé par deux transitions : de l'état *Current* à l'état *Droite*, *Haut* ou *Bas*, et retour à l'état *Current*, ce qui génère au moins 2 états supplémentaires dans le graphe d'états parcouru par le model-checking. Malgré ces remarques, il convient de souligner que c'est l'ordre de grandeur de la complexité qui nous intéresse ici.

Dans les tables 6.1 et 6.2, nous présentons les résultats de vérification de la propriété d'atteignabilité totale bord à bord sur le modèle MRD. L'algorithme de référence vérifie toujours w^2 propriétés d'atteignabilité sur la grille.

Nous avons construit une base de tests formée par des grilles de taille 4x16 (taille I) et des grilles de taille 4x32 (taille II). Pour générer ces grilles, nous avons utilisé une méthode ad-hoc. Le cas 'allA' qui est le cas où l'action A est affectée à toutes les cases de la grille est le pire cas de complexité car tous les chemins sont possibles. Nous avons généré une base de test composée de 18 grilles aléatoires pour lesquelles Π n'est pas vérifiée dont le cas 'allB', et 18 grilles pour lesquelles Π est vérifiée dont le cas 'allA'. Nous appelons chaque cas de la base d'essai un *scénario*.

Les résultats expérimentaux présentés dans la table 6.1 ont été obtenus avec le model checker UppAal en mode Breadth First Search (BFS). Les deux modes de recherche possibles sont : Breadth First Search (BFS - en largeur d'abord) et Depth First Search (DFS - en profondeur d'abord). Dans notre cas, la propriété est vérifiée sur une feuille du graphe d'états. Si le mode de recherche est DFS, la case du bord droit correspondant à la recherche du chemin peut se trouver sur la première branche du graphe d'états (très peu d'états seront alors générés) comme elle peut se trouver sur la dernière branche (tout l'espace d'état est alors exploré). Si le mode est BSF, la première solution à la requête d'atteignabilité, qui nécessite d'atteindre le bord droit de la grille, sera toujours éloignée de la racine du graphe d'états. Le mode de recherche BFS a été choisi car il devrait présenter moins de variation de performance entre les requêtes pour une grille donnée.

Afin d'analyser les résultats expérimentaux obtenus, il est utile de considérer les points suivants :

- L'algorithme de référence vérifie toujours 4^2 propriétés d'atteignabilité sur des grilles de taille 4×16 et 4×32. En effet, pour l'algorithme de référence, le nombre de propriétés d'atteignabilité à vérifier ne dépend pas de la longueur de la grille (nombre de colonnes) mais seulement de sa largeur (nombre de lignes).

- Le model checker d'UppAal utilise une technique de recherche à la volée (*on the fly*). Il s'arrête donc dès qu'une solution est trouvée. C'est pourquoi il ne génère pas tout le graphe d'états lorsqu'il existe une solution. Néanmoins, dans l'ordre BFS, la solution est éloignée de la racine du graphe d'états car elle se trouve uniquement au bord droit de la grille.

- La taille de l'espace d'états dépend de la distribution des actions dans la grille. Le cas "allA" est théoriquement le scénario le plus gourmand en mémoire car tous les chemins sont possibles.

- Au moment de la vérification, UppAal alloue un nombre minimum de blocs de mémoire, ce qui constitue le minimum de mémoire consommée par le processus d'analyse, même si la vérification effective pourrait consommer moins de mémoire (ce qui est le cas pour certains des cas d'études analysés) [5].

5. `https://www.it.uu.se/research/group/darts/uppaal/press/uppaal-3.4.shtml`

Le processus d'analyse et certaines spécificités de l'outil et/ou de la démarche ayant été rappelés nous allons maintenant présenter et analyser les résultats expérimentaux obtenus en nous intéressant d'abord à la quantité de mémoire utilisée.

A) Complexité expérimentale en espace mémoire utilisée

La table 6.1 présente les résultats expérimentaux obtenus sur nos benchmarks avec un PC doté d'un processeur Intel E5 cadencé à 3.2 GHz et doté de 256 Gbits de RAM.

Dans cette table, les 3 premières colonnes présentent les caractéristiques des scénarios utilisées, à savoir les dimensions de la grille analysée, la classe du scénario (Π vérifiée ou non) et la référence du scénario (taille.n°). La 4ème colonne indique la quantité de mémoire, exprimée en KBytes, utilisée pour résoudre le problème d'atteignabilité en faisant appel à l'algorithme de référence.

D'après la table, on remarque que les besoins en mémoire ne sont pas uniformes entre les différents scénarios de la base de test. Si on exclue le cas extrême allA, la mémoire requise pour la vérification de la propriété d'atteignabilité se situe entre 76 KB et 37 GB. Les résultats montrent que la complexité est exponentielle en fonction de la taille de la grille. Le passage de la taille I (grille 16×4) à la taille II (grille 32×4) s'accompagne d'une grande augmentation des besoins en mémoire. Dans la plupart des cas où la propriété Π n'est pas vérifiée, comme les scénarios I.allB, I.1, I.3, I.5, I.6, I.8, I.10, II.allB, II.1 et II.2 les besoins en mémoire pour la résolution de la propriété Π sont faibles, voire négligeables. Notre explication est que la complexité dépend du nombre de chemins générés au moment de la vérification de la propriété, ainsi que de la taille de ces chemins. Cela dépend de la répartition des actions dans la grille. Ainsi, si le nombre de chemins possibles est petit, le graphe d'état généré par la vérification est simple donc la vérification nécessite très peu de mémoire. Cette explication a été confortée par des tests avec des grilles spécifiques. Pour les scénarios I.1, II.1, I.3, I.5, I.6, I.8, I.10, les besoins en mémoire sont les mêmes, ceci provient du fait qu'au moment de la vérification, UppAal alloue un ensemble minimum de blocs mémoire de 76KB. Ces scénarios correspondent au cas où aucun chemin complet n'existe dans la grille entre la gauche et la droite. Les autres scénarios pour lesquels Π n'est pas vérifiée, mais qui ont consommé plus de mémoire, contiennent des chemins plus ou moins complexes, mais qui ne permettent pas de vérifier la propriété d'atteignabilité totale.

Le cas allA est le pire cas. Il permet de tester les limites physiques de résolution du problème d'atteignabilité avec notre ordinateur. Pour le scénario II.allA par exemple, au bout d'une semaine, la mémoire consommée avait atteint 125 GB. Nous avons continué les tests jusqu'à 33 jours et la mémoire consommée avait atteint 130 GB sachant que la vérification n'était pas terminée et que nous avions atteint seulement la moitié de la capacité mémoire de notre ordinateur. On s'expose donc ici à une autre complexité du problème : le temps de résolution, que nous abordons ci-dessous.

TABLE 6.1 – Les besoins en Mémoire (M) pour la résolution du problème d'atteignabilité bord à bord avec l'algorithme de Référence

Grille	Classe	Scénarios	M (KB)
Taille I	Π non vérifiée	I.1, I.3, I.5, I.6, I.8, I.10, I.allB	76
		I.2	8 388
		I.4	6 276
		I.7	6 548
		I.9	4 088
	Π vérifiée	I.1	76
		I.2	6 416
		I.3	8 320
		I.4	6 804
		I.5	8 600
		I.6	12 280
		I.7	13 600
		I.8	14 920
		I.9	21 456
		I.allA	1 835 732
Taille II	Π non vérifiée	II.allB, II.1, II.2	76
		II.3	9 000
		II.4	12 420
		II.5	27 564
		II.6	87 144
		II.7	344 572
	Π vérifiée	II.1	598 656
		II.2	5 473 520
		II.3	2 280 964
		II.4	10 472 500
		II.5	17 548 756
		II.6	9 033 336
		II.7	36 800 592
		II.allA	*>130 GB*

B) Complexité expérimentale en temps : Durée d'exécution

La table 6.2 présente les temps d'exécution enregistrés sur nos benchmarks. Les 3 premières colonnes présentent les caractéristiques des scénarios utilisées. La 4ème colonne indique la durée d'exécution nécessaire à la vérification de toutes les propriétés du problème exprimée en secondes.

Les résultats que nous obtenons sont très hétérogènes, mais reflètent globalement les résultats en mémoire. On observe que TE varie de 2s à 15000 secondes. Pour

TABLE 6.2 – Les besoins en Temps d'Exécution (TE) pour la résolution du problème d'atteignabilité bord à bord avec l'algorithme de Référence

Grille	Classe	Scénarios	TE (s)
Taille I	Π non vérifiée	I.1, I.3- I.6, I.8- I.10, I.allB	2
		I.2, 1.7	3
	Π vérifiée	I.1 - I.5	2
		I.6, I.9	6
		I.7, I.8	7
		I.allA	358
Taille II	Π non vérifiée	II.allB, II.1, II.2	2
		I.3	3
		I.4	8
		I.5	18
		I.6	59
		I.7	239
	Π vérifiée	II.1	315
		I.2	2 235
		I.3	1 076
		I.4	5 372
		I.5	7 117
		I.6	5 209
		I.7	15 308
		II.allA	–

les scénarios de taille I et quelque soit la classe de scénario, le TE est négligeable (<10s) sauf pour le cas I.allA. Les variations du temps de calcul dépendent de la répartition des actions dans la grille. Lorsque la grille croit en dimensions (taille II), TE augmente de façon exponentielle. Pour le scénario II.1 de la classe "Π *est vérifiée*", le TE est 315s qui est beaucoup plus petit que le TE des autres scénarios de la même classe. Une analyse plus approfondie permet d'expliquer les raisons induisant ce comportement. En effet, le nombre de chemins généré pour ce scénario est petit donc la vérification n'est pas complexe. Comme déjà expliqué, le cas II.allA est le pire cas et il permet d'atteindre les limites pratiques de la résolution avec un ordinateur personnel, au niveau temps de résolution.

En conclusion, la vérification de notre propriété d'atteignabilité totale bord à bord avec l'algorithme de référence devient de plus en plus complexe en mémoire et en temps d'exécution lorsque la taille de la grille croit en dimensions. Un problème d'explosion combinatoire se produit lorsque la grille est de taille II et que toutes les actions de la grille sont à A. Pour faire face à ce problème, une solution sera proposée au chapitre 9.

6.6 Conclusion

Dans ce chapitre, nous avons étudié comment les techniques de model-checking pouvaient être appliquées pour traiter les propriétés spatiales des robots mobiles dans un environnement ouvert. L'environnement agricole est représenté comme une grille spatiale rectangulaire régulière à deux dimensions (2D). Dans chaque case de la grille, le robot effectue une seule action. Le robot est soumis à plusieurs contraintes : des contraintes de déplacement, des contraintes de visite et des contraintes de précédence entre les actions. Le fonctionnement du robot a été modélisé sous la forme d'un réseau d'automates temporisés et le modèle conçu a été appelé Modèle Région de Déplacement (MRD). La propriété de vérification étudiée est celle de l'atteignabilité de toutes les cases d'un bord de la grille à toutes les case de l'autre, appelée propriété d'atteignabilité totale bord à bord. Nous avons décrit l'algorithme de référence qui permet de vérifier cette propriété sur le modèle MRD. Nous avons présenté les performances de l'algorithme de référence en nous concentrant sur l'impact en besoins en mémoire et en temps d'exécution. La complexité de l'algorithme dépend de la taille de la grille, de la répartition des actions dans la grille et également du choix de l'ordre de recherche. Au moment de la vérification de la propriété sur une grille de taille 32 colonnes et 4 lignes, un problème d'explosion combinatoire se produit : l'exploration exhaustive de l'espace d'états du problème conduit à la génération d'un trop grand nombre de mouvements possibles. Pour faire face à ce problème, une approche de décomposition sera proposée dans la partie 3 de ce manuscrit.

Conclusion générale de la partie 2

Dans la partie II de ce manuscrit, nous avons vu 3 cas d'une application des techniques de model-checking à des problèmes issus de l'agriculture de précision. Les problèmes agricoles traités sont : la pulvérisation sur un rang de vigne, la récolte sélective sur un champ et le parcours d'un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques.

Il s'agit dans les trois cas de vérifier l'aptitude de l'équipement agricole à respecter les contraintes et les objectifs de la tâche à effectuer, et, dans le même temps, de réaliser une synthèse de commande pour l'équipement. Les contraintes sont de deux types : 1) celles liées à la dynamique de la machine agricole, et 2) celles issues de l'agriculture de précision.

Pour chacun de ces problèmes, nous avons établi une méthode spécifique permettant de répondre au problème traité, en nous appuyant sur les 3 étapes du model checking : (i) modélisation formelle du comportement du système, (ii) représentation des propriétés à vérifier sur le système pour sa validation, (iii) enfin vérification des propriétés par model checking.

Pour la pulvérisation sur un rang de vigne, nous avons proposé la méthode AMPS (Automate Modeling for Precision Spraying). Cette méthode est basée sur le formalisme des Automates Temporisés de Coût (ATC) et sur l'outil de vérification UppAal-CORA. Elle prend en entrée des données terrain issues d'un scan LiDAR 2D de la canopée et, en considérant la dynamique du pulvérisateur (ouverture et fermeture des buses), elle calcule la séquence de contrôle optimale permettant de minimiser la quantité de produit pulvérisé. Nous avons montré sur un exemple concret de rang de vignoble que notre méthodologie AMPS donnait des résultats prometteur, mais qu'elle était rapidement limitée en pratique par un problème d'explosion combinatoire.

Pour la récolte sélective sur un champ, nous avons établi une approche, également basée sur le formalisme ATC et l'outil UppAal-CORA, permettant de calculer une logistique de récolte optimale à partir d'une carte de préconisation spatiale qui distingue des zones de qualité, avec deux qualités de raisins. Ces zones sont identifiées indépendemment des rangs et donc chaque rang peut avoir des raisins des deux qualités. L'objectif de la récolte sélective est de calculer une logistique de récolte optimale permettant de récolter au moins une quantité $Rmin$ de raisins de meilleure

qualité tout en respectant la carte de préconisation et en minimisant la durée totale de la récolte. Nous avons montré sur un exemple concret de parcelle de vigne que l'approche proposée permet de trouver la logistique optimale pour des parcelles formée par 12 rangs, mais qu'elle est ensuite limitée en pratique par l'explosion combinatoire.

Pour le parcours d'un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques, nous avons étudié le cas d'un robot qui se déplace dans un environnement ouvert, représenté comme une grille spatiale. Le robot effectue une action spécifique dans chaque case de la grille, et il est soumis à plusieurs contraintes limitant ses déplacements. La mission du robot est formulée comme une propriété d'atteignabilité particulière (*atteignabilité totale bord à bord*). Comme pour les deux applications précédentes, la méthode donne des résultats intéressants mais au moment de la vérification de la mission du robot, un problème d'explosion combinatoire se produit lorsque la grille est de taille significative.

En conclusion, nous avons montré que des approches basées sur le model-checking offre des opportunités très intéressantes pour la vérification et la génération de commandes optimisées pour des problèmes d'agriculture de précision. Cependant, appliqué d'une façon directe, le processus de vérification conduit à des problèmes d'explosion combinatoire, mis en évidence sur les 3 applications étudiées. Les deux premières applications (la pulvérisation et la récolte) se basent sur l'utilisation de l'outil UppAal-CORA qui est construit pour une architecture 32 bits limitant ainsi l'utilisation de la mémoire RAM à 4 Go au moment de la vérification. Cependant, la complexité combinatoire du model-checking est également une réalité théorique, intrinsèque à son exhaustivité et à la complexité des problème étudiés, qui ne serait pas levée dans ces deux applications par la simple mise à disposition d'un exécutable 64 bits.

Pour dépasser les problèmes d'explosion combinatoire liés à une application directe des techniques de model-checking et pour aller plus loin dans l'exploration des possibilités offertes par ces techniques pour l'agriculture de précision, la partie 3 de ce manuscrit explore la possibilité de décomposer la résolution du problème. Ce principe de décomposition est mis en oeuvre sur deux des trois exemples étudiés.

Troisième partie

Décomposer le model-checking

7 Introduction et principes de décomposition

Comme illustré dans la partie précédente (partie II) avec plusieurs exemples, un problème d'explosion combinatoire se produit au moment de la vérification dans de nombreux problèmes réalistes. Ce problème est la principale limite des techniques de model-checking. La méthode de model-checking est intrinsèquement complexe à cause de son approche d'énumération exhaustive. Pour traiter des problèmes réels, souvent complexes, il est donc nécessaire de trouver des solutions pour résoudre, ou tout du moins gérer, l'explosion combinatoire. Nous avons parcouru rapidement en section 2.5.3 quelques solutions existantes dans la littérature à savoir le model-checking symbolique par les BDD, les techniques d'ordre partiel, la technique d'exploitation des symétries et la technique de vérification à la volée. Or toutes ces solutions sont déjà implémentées dans l'outil de model-checking UppAal. On peut trouver dans [6] la description des développements visant à améliorer l'efficacité de l'outil UppAal, en particulier le développement de la nouvelle structure de données CDD (Clock Difference Diagrams) qui est une généralisation du BDD, et le développement de techniques d'approximation et de réduction d'ordre partiel. D'après [13, 71], la technique de réduction de la symétrie est une technique très connue, visant à réduire les besoins en ressources pour les algorithmes de model-checking. Elle a été implémentée avec succès dans plusieurs outils de model-checking (SMV, Spin, etc) et elle est également ajoutée dans l'outil UppAal pour atténuer le problème de l'explosion de l'espace d'états.

Nous avons donc besoin d'une nouvelle approche permettant de réduire l'impact de ce problème d'explosion combinatoire, ce qui permettra de traiter des modèles de plus grande taille et ainsi de résoudre des problèmes réalistes rencontrés en agriculture.

L'approche que nous allons proposer est inspirée du travail de Koo et Mishra [80] qui présente une technique efficace de génération de tests basée sur la décomposition de la conception (design decomposition) et des propriétés pour la validation fonctionnelle de processeurs en pipeline. [80] définit 4 scénarios possibles pour générer les tests en utilisant potentiellement une double décomposition :

1. Le premier scénario consiste à vérifier la propriété globale sur le modèle global du processeur "Processor Model" ; il s'agit là du model-checking traditionnel.

2. Le deuxième scénario consiste à décomposer seulement la propriété en des sous-propriétés et vérifier les sous-propriétés sur le modèle global.

3. Le troisième scénario consiste à décomposer seulement le modèle en des sous-modèles et vérifier la propriété globale sur les sous-modèles.

4. Le quatrième scénario est la double décomposition.

La méthode de double décomposition implémentée dans [80] est un peu particulière. L'objectif de cette double décomposition est la génération des tests. On rappelle que les outils de model-checking peuvent fournir un contre exemple lorsque la propriété est non vérifiée. Dans [80], le modèle est décomposé en composants de manière à ce qu'il y ait très peu d'interaction entre les composants partitionnés. Dans le cas du processeur en pipeline, la décomposition du modèle est naturelle. En effet, le processeur en pipeline partitionné est considéré comme un graphe où les nœuds sont constitués d'unités de calcul ou de stockage (lecture - *fetch*, décodage - *decode*, mémoire - *memory*, registre - *register*, etc). Pour la décomposition de la vérification, la technique employée dans [80] est de générer d'abord la négation de la propriété globale et ensuite de décomposer la propriété déjà sous format de négation en des sous-propriétés. La vérification des sous-propriétés sur les sous-modèles permet de générer des contre-exemples. La fusion de ces contre-exemples produit des scénarios de tests.

Nous reprenons l'idée de la double décomposition pour nos travaux mais en l'adaptant et la repensant pour notre contexte. Il s'agit pour nous également de faire deux types de décomposition : une décomposition en modélisation, que l'on retrouve en littérature pour la conception modulaire et la vérification de composants, et une décomposition de la vérification elle-même.

Nous proposons ici une nouvelle approche de décomposition du processus de vérification par model-checking pour l'agriculture de précision. La représentation spatialisée de nos problèmes va constituer la clé de voûte de notre proposition, et la technique proposée va se baser sur la particularité de spatialisation (1D, 1D+, 2D) des problèmes traités qui permet plus aisément de décomposer le problème.

Du point de vue de l'arbre d'exploration du model checking (le graphe d'états), la méthodologie de décomposition que nous proposons consiste à chercher des états considérés comme des points de repère ou des points de passage obligés. Ces points partagent ou satisfont des propriétés spécifiques qui différent d'une application à une autre.

Le principe générique de cette méthodologie de décomposition est constitué de trois phases :

Décomposition de la modélisation : partitionner le modèle en n sous-modèles de sorte à avoir une zone de recouvrement entre 2 sous-modèles consécutifs. Cette zone de recouvrement pourrait être un bloc (cas du pulvérisation) ou une colonne d'une grille (cas de grille 2D). La méthodologie de décomposition consiste à chercher des états considérés comme des points de repère ou des points de passage obligés.

Décomposition de la vérification : décomposer la résolution de la propriété à vérifier sur le modèle global en la résolution de propriétés sur les sous-modèles. Les propriétés à vérifier dans les requêtes sur les sous-modèles peuvent être différentes de la propriété à vérifier sur le modèle global, mais la réponse finale doit rester cohérente.

Analyse de la décomposition : prouver que la vérification des propriétés sur les sous-modèles permet la vérification de la propriété initiale sur le modèle global. Il s'agira ici soit d'essayer de prouver que la vérification après décomposition vérifie exactement les mêmes propriétés que la vérification initiale, soit de déterminer les limites de cette vérification.

Dans la suite, le principe générique de décomposition exposé ici sera décliné sur deux exemples caractéristiques que nous avons déjà étudiés : l'optimisation de la pulvérisation de précision et la question d'atteignabilité sur une grille 2D.

8 Décomposition et pulvérisation de précision

Sommaire

Dans le chapitre 4, nous avons présenté la méthode AMPS qui permet la détermination, grâce au model checking, de la séquence de contrôle optimale d'un pulvérisateur à 3 mains, adaptant la dose de pesticide projetée sur un rang de vignoble en fonction de la densité de végétation observée. Elle se décompose en 3 étapes :

1. Définir une fonction statique qui associe à chaque état de végétation possible deux commandes différentes de pulvérisation (C_{best}, C_{alt}).

2. Traiter les données LiDAR d'un rang de végétation réel pour obtenir C_{best}, C_{alt} et la durée de pulvérisation de chaque bloc homogène de végétation pour le rang étudié.

3. Calculer la séquence de commande optimale du pulvérisateur à l'aide d'un modèle basé sur des automates temporisés de coût et sur l'outil de vérification UppAal-CORA.

Lors de l'application de la méthode AMPS sur des données réelles reflétant l'état de végétation d'un rang de vigne de 180 m de long, nous avons été confrontés au phénomène classique en model-checking d'explosion combinatoire. C'est pourquoi la méthode proposée n'a pu être appliquée dans le chapitre 4 que sur 40 blocs alors que 262 blocs de végétation ont été identifiés pour ce rang.

Pour pouvoir adresser des problèmes de dimensions réelles, nous décrivons dans ce chapitre une méthodologie de décomposition basée sur le principe générique de décomposition vu plus haut. La décomposition se base ici sur la représentation spatiale 1D du rang de vigne en blocs de végétation. Cette méthodologie a été développée et appliquée sur notre exemple pour résoudre le problème d'optimisation de la pulvérisation de précision et vérifier la propriété d'atteignabilité sur tout le rang de vigne.

8.1 Méthodologie de décomposition

En nous appuyant sur les principes généraux de décomposition que nous avons énoncés en introduction de cette partie, nous allons maintenant décliner cette méthodologie à l'exemple de l'optimisation de la pulvérisation de précision pour les vignobles (voir chapitre 4).

Nous avons basé notre décomposition sur les blocs où la commande est connue a priori car unique. Commençons donc par nous intéresser à l'étape de décomposition de la modélisation.

8.1.1 Décomposition de la modélisation

La décomposition de la modélisation que nous proposons pour ce problème se divise en deux étapes qui consistent à :

1. Former des parties : décomposer le rang R en n parties du rang (P_1, ..., P_n).

2. Générer des sous-modèles : générer le sous-modèle associé à chaque partie.

8.1.1.1 Décomposition grâce aux blocs "trou de végétation"

Comme nous l'avons vu dans le chapitre 4, le choix d'utiliser C_{best} ou C_{alt} dans une commande optimale pour un bloc donné peut être influencée par les commandes possibles des blocs qui le suivent et le précèdent. Cependant, la présence d'un trou entre 2 blocs casse cette notion de dépendance, car elle engendre obligatoirement la commande de fermeture de toutes les buses. Nous appellerons état stable qui a cet effet de casser la dépendance entre la suite de commandes avant l'état stable et la suite de commande après cet état. Ainsi, la présence de trous dans un rang de vigne représente une situation particulière à considérer lors de la décomposition de notre

problème. Dans un premier temps, nous étudions donc la décomposition en fonction de la présence des trous de végétation.

Ainsi, un rang R est décomposé en n parties $\{P_1, ..., P_n\}$, c'est-à-dire en n sous-ensemble de blocs. Les parties sont définies de façon à avoir entre elles un bloc de recouvrement de type *"Trou"* qui appartiendra aux deux parties. Ainsi, une partie est un ensemble de blocs qui commence par un bloc de type *"Trou"* ou par le premier bloc du rang, et qui se termine par un bloc de type *"Trou"* ou par le dernier bloc du rang. La figure 8.1 illustre, par un exemple, la formation des parties. Le rang R est formé par $m + 1$ blocs. Les blocs $\{b_i, b_j, b_k, b_l\}$ sont des blocs de type *"Trou"*. La partie P_1 est composée de la succession des blocs du bloc b_0 de début de rang au bloc b_i. La partie P_2 est la succession des blocs du bloc b_i au bloc b_j. Le bloc b_i est le bloc de recouvrement (BR) entre P_1 et P_2. On poursuit le même raisonnement pour les autres parties jusqu'à P_n.

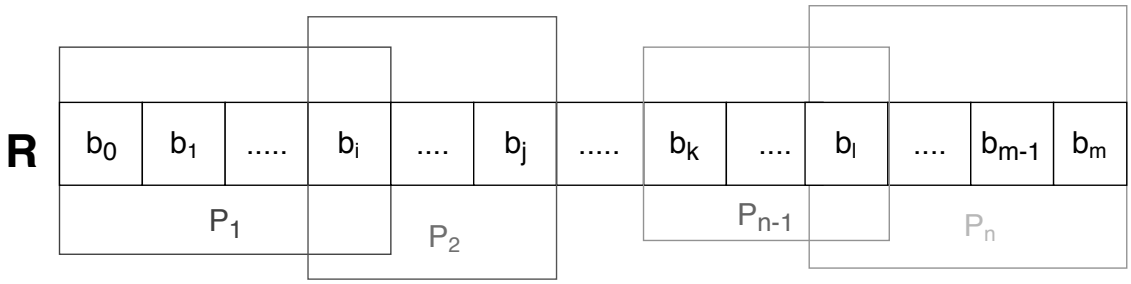


FIGURE 8.1 – Décomposition du rang R en n parties

Cependant cette première approche de décomposition générique et guidée par les trous peut se révéler insuffisante pour réduire significativement la complexité du problème si il existe une trop grande séquence de blocs sans trou. Nous proposons donc dans le paragraphe suivant une approche complémentaire, dite approche de décomposition par injection, permettant de réduire la complexité de notre problème.

8.1.1.2 Décomposition par injection

L'idée générale de décomposition peut alors être adaptée en considérant que la décomposition peut se faire pour n'importe quel bloc de recouvrement si sa commande est connue a priori de façon déterministe, ou si cette commande n'a pas d'influence sur notre critère de vérification. Dans notre cas, cela signifie que les blocs pour lesquels $C_{best} = C_{alt}$ peuvent également être considérés comme des blocs de recouvrement potentiels. Par contre, contrairement aux blocs trous, un $C_{best} = C_{alt}$ n'oblige pas à fermer toutes les buses : les temps de fermeture/ouverture des buses restent dépendant des commandes antérieures et postérieures à ce bloc.

Prenons comme exemple un ensemble de 5 blocs (sans trou) que nous nommerons b_i, b_{i+1}, b_{i+2}, b_{i+3} et b_{i+4}, le bloc b_{i+2} étant un bloc où $C_{best} = C_{alt}$. La figure 8.2

présente l'ensemble des commandes possibles. B_iC_{best} désigne la commande C_{best} du bloc b_i et B_iC_{alt} désigne la commande C_{alt} du bloc b_i. Le même raisonnement s'applique pour le reste des blocs. On remarque que le bloc b_{i+2} constitue un point de passage obligé quelque soit les commandes C_{best} et C_{alt} qui ont été choisies pour les autres blocs, nous allons donc baser notre décomposition sur ce bloc.

Soit $P2$ la partie qui regroupe tous les blocs. On peut supposer que $P2$ est le résultat d'une première décomposition basée sur les trous de végétation, mais que cette partie est encore trop grosse et qu'on désire encore la décomposer. Soit $P1$ la partie qui regroupe les 3 premiers blocs. Pour $P1$, il existe 4 séquences de commandes possibles et pour $P2$, il en existe 16. Soit $SO(P1)$ la séquence optimale pour $P1$, $SO(P2)$ la séquence optimale pour $P2$ et $(SO(P2))[P1]$ est la sélection de la séquence optimale des 3 premiers blocs formant $P1$.

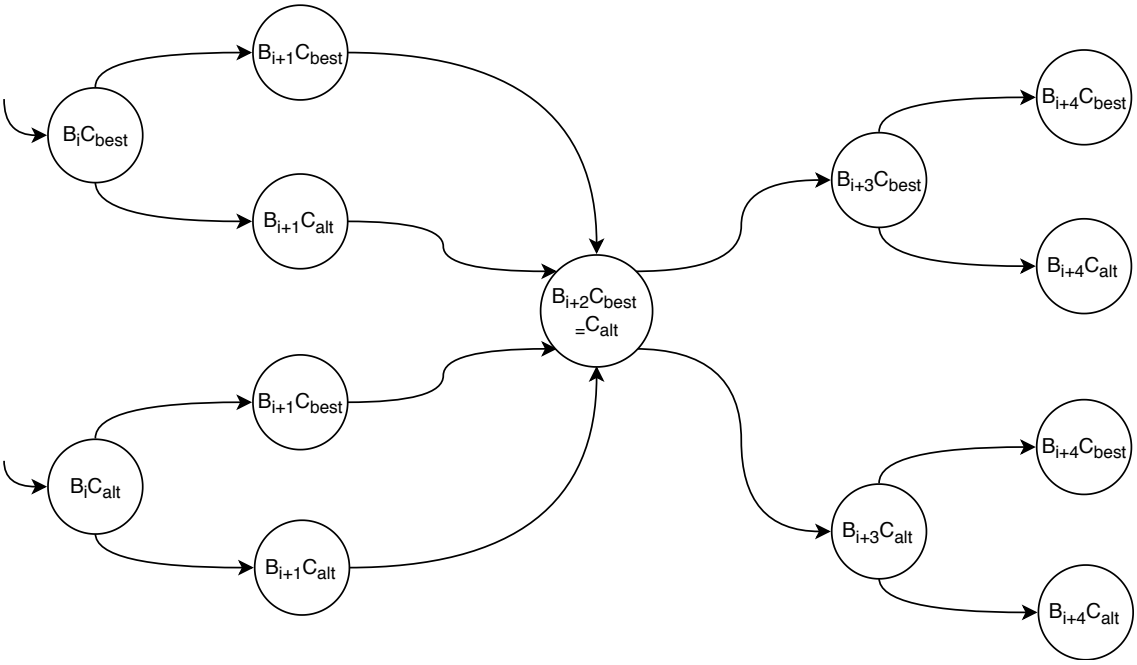


FIGURE 8.2 – Exemple d'un bloc où $C_{best} = C_{alt}$

Dans notre exemple, contrairement au cas des trous de végétation, pour b_{i+2} rien n'oblige à fermer toutes les buses, donc le coût optimal trouvé pour $P2[P1]$ est différent au coût optimal trouvé pour $P1$ (fermeture des buses pour $P1$) mais la séquence optimale $SO(P2)[P1]$ trouvée pour $P2[P1]$ est la même que la séquence optimale $SO(P1)$ trouvée pour $P1$.
On en retient que du point de vue ouverture et fermeture de buses, il y a une dépendance en ce qui concerne le coût optimal mais il n'y a pas de dépendance en ce qui concerne la séquence optimale. L'idée de décomposition par injection est de mémoriser la séquence optimale $SO(P1)$, et d'*injecter* cette solution dans la partie

$P2$. Ainsi, dans la partie $P2$, les blocs de $P1$ auront leur commande optimale imposée ce qui permet de supprimer la complexité liée au choix des commandes pour chaque bloc de $P1$. On appelle cette méthode de décomposition *décomposition par injection*. Cette méthode ne nécessite pas de modifier les automates du modèle, simplement les tableaux $Commande[1]$ et $Commande[2]$ spécifiant les commandes possibles pour les rangs.

La question qui se pose maintenant est comment utiliser ces 2 critères de décomposition ? Sont-elles exclusives ou complémentaires ? La réponse à ces questions est l'objet de la section suivante qui décrit la stratégie de partitionnement employée.

8.1.1.3 Stratégie de décomposition employée

Les deux approches de décomposition, décomposition orientée trou et décomposition par injection, sont deux approches complémentaires. Il serait possible d'utiliser seulement l'approche par injection mais cette approche nécessite de mémoriser explicitement les commandes ce qui complexifie la résolution et consomme de la mémoire. Nous utiliserons donc cette solution en deuxième recours que lorsque la décomposition orientée trou ne suffit pas.
Ainsi, la stratégie de décomposition mise en œuvre est la suivante :

1. Décomposition orientée trou sur tout le rang,
2. Appel au processus de vérification pour les parties formées,
3. En cas de problème d'explosion combinatoire, appel à la décomposition par injection pour les parties étant encore de trop grande dimension.

Dans la section suivante, les modifications à apporter sur le modèle pour générer les sous-modèles sont précisées.

8.1.1.4 Modification du modèle et génération des sous-modèles

Au sein du modèle, la décomposition du modèle initial en sous-modèles se traduira de la façon suivante :

- Variables globales : diminution du nombre de blocs pour chaque sous-modèle à la valeur du nombre de blocs formant la partie. Ainsi les dimensions de tous les tableaux stockant les informations de ces blocs seront réduites (typiquement les tableaux $Commande[1]$, $commande[2]$, $tempo\_bloc$ et $Choice$).
- Aucun changement dans les automates

Même si cela n'a pas été le cas pour ces travaux de thèse, faute de temps, la décomposition de la modélisation peut aisément être automatisée. Le processus pratique de génération des sous-modèles s'appuie sur un fichier texte `data.txt`, obtenu

à la fin de l'étape 2 de AMPS et décrivant : le nombre de blocs formant le rang, la durée de chaque bloc *tempo_bloc*, les commandes C_{best}, C_{alt} pour chaque bloc. La génération des sous-modèles décompose, en fonction de la présence des trous, le contenu du fichier data.txt en n fichiers `data_i.txt` avec $i \in \{1, .., n\}$. Le fichier `data_1.txt` contient le nombre de blocs formant la partie 1, la durée, les commandes C_{best}, C_{alt} et pour la décomposition par injection la commande choisie $Choice$ pour chaque bloc de la partie 1. En utilisant ce fichier et le fichier .xml décrivant le modèle initial correspondant à tout le rang, on génère le fichier .xml du sous-modèle correspondant à la partie P1 ($SMP1$: Sous-Modèle de la partie P1). On applique ensuite le même processus pour générer le sous-modèle de chaque partie formant le rang.

Après avoir présenté la décomposition en modélisation, abordons maintenant la décomposition en vérification.

8.1.2 Décomposition de la vérification

La propriété permettant d'obtenir la séquence de contrôle optimale du point de vue du coût à appliquer sur tout le rang est la propriété 3 décrite dans la section 4.3.4 [1] :

$$E <> currentBlock == nb\_block - 1 \text{ and } ProcessVeg.end \quad \text{(Propriété 3)}$$

La fonction de coût dans notre modèle de pulvérisation calcule la quantité totale de produit pulvérisé. La vérification de la propriété 3 en mode optimisation (CORA) permet d'obtenir la séquence de commandes C_{best} ou C_{alt} pour chaque bloc en optimisant le coût global, c'est à dire la quantité minimale de produit nécessaire pour traiter le rang tout en assurant une protection suffisante.

8.1.2.1 Décomposition de la vérification pour la décomposition par trou

La décomposition de la vérification consiste à vérifier la propriété 3 sur chacune des parties du rang. Il suffit simplement d'appliquer l'approche de vérification présentée au chapitre 4 à chacun des sous-modèles identifiés. Dans le modèle d'une partie, *nb_block* devient le nombre de blocs formant la partie et $ProcessVeg.end$ est atteint quand on finit de traiter tous les blocs de cette partie. Vérifier la propriété 3 sur le sous-modèle $SMPi$ permet de calculer la Séquence Optimale (SO_i) à appliquer sur cette partie Pi et permet également de calculer le Coût Optimal CO_i, c'est à dire la quantité minimale de produit nécessaire pour pulvériser cette partie du rang.

La séquence optimale complète est obtenue en faisant l'union des séquences optimales des parties, et la somme des coûts optimaux des parties permet d'obtenir le coût optimal global (cela sera prouvé section 8.1.3.3).

1. On rappelle que *currentBlock* est le bloc courant que le pulvérisateur est en train de pulvériser et *nb_block* est le nombre de blocs formant le rang.

8.1.2.2 Décomposition de la vérification pour la décomposition par injection

Pour des raisons liées à la simplification du mécanisme de modification des modèles, nous allons appliquer une méthode différente pour la décomposition de la vérification pour lorsque le principe de décomposition est par injection.

Soit P une partie que nous voulons décomposer par injection en trois parties $P1$, $P2$ et $P3$. La notation $SMPX^{Y-Z}$ avec X,Y et Z des numéros des parties signifie que les parties en exposant, les parties Y et Z, sont des parties injectées :leur commande optimale est imposée (déjà calculée). La commande optimale pour la partie X, en position normale, n'est pas encore déterminée. Nous allons travailler sur 3 modèles :

- Un modèle $SMP1$ qui contient uniquement les blocs de $P1$.
- Un modèle $SMP2^1$ qui contient les blocs de $P1$ et $P2$. La différence réside dans la possibilité de choix des commandes : le choix n'a pas encore été fait entre C_{best} et C_{alt} pour les blocs correspondant à la partie $P2$. Les blocs de la partie 1, eux, ont leur commande optimale imposée : c'est ce que nous appelons l'injection. L'injection permet de supprimer la complexité liée au choix des commandes pour chaque bloc de $P1$.
- Un modèle $SMP3^{1-2}$ qui contient les blocs de $P1$ et $P2$ avec leur commande optimale injectée, et les blocs de $P3$ avec leur commande au choix.

Ainsi, la décomposition de la vérification de la propriété 3 sur la partie P, lorsque P est décomposée par injection, s'effectue en 3 étapes :

- Vérification de la propriété 3 sur la partie $P1$ seule, et récupération de la séquence optimale et du coût optimal.
- Modification du modèle $SMP2^1$ pour injecter la séquence optimale des blocs de $P1$, puis vérification de la propriété 3 sur ce nouveau modèle ; récupération de la séquence et du coût optimaux pour $P1 + P2$.
- Modification du modèle $SMP3^{1-2}$ pour injecter la séquence optimale des blocs de $P1$ et $P2$, et récupération de la séquence et du coût optimaux pour $P1 + P2 + P3$.

La propriété recherchée a donc été vérifiée pour chacun des sous-modèles si, bien évidemment, la complexité combinatoire a été suffisamment réduite. Il nous faut maintenant démontrer que la composition des séquences optimales calculées pour chacun des sous-modèles permet effectivement d'obtenir la séquence globale optimale, c'est-à-dire à coût minimal.

8.1.3 Analyse de la décomposition

Nous prouvons ici que si la propriété 3 est vérifiée sur tous les sous-modèles $SMPi$ avec i ∈ { 1, .., n } alors la propriété 3 est vérifiée sur le modèle global PTA_AMPS.

8.1.3.1 Vocabulaire

Nous allons tout d'abord introduire des notations formelles relatives au rang R. Soient :

- m le nombre de blocs dans R,
- cmd_k, la commande choisie pour le bloc k du rang dans la solution optimale,
- co_k, le coût (ici en quantité de produit) correspondant à cmd_k,
- On peut décomposer ce coût de façon plus précise, comme cela avait été fait de façon informelle dans la section 4.3.1.3, et pour $k \in \{1, n\}$ n étant le nombre de rangs :

$$co_k = Ferm_k(cmd_{k-1}) + b_k(cmd_k) + Ouv_k(cmd_{k+1})$$

avec :

  - X une buse du pulvérisateur : $X \in \{LH, CH, HH\}$
  - $Ferm_k(cmd_{k-1})$ représente la fermeture, au début du rang k, des buses impliquées dans la commande du bloc précédent, et inutiles dans k :

$$Ferm_k(cmd_{k-1}) = \sum_{X \in cmd_{k-1} \wedge X \notin cmd_k} Ferm(X)$$

  - La consommation pendant le rang k des buses utilisées pour sa commande :

$$b_k(cmd_k) = \sum_{X \in cmd_k} b_k(X)$$

  - $Ouv_k(cmd_{k+1})$ représente l'ouverture, à la fin du rang k, des buses nécessaires pour la commande du bloc suivant, et non utilisées dans k :

$$Ouv_k(cmd_{k+1}) = \sum_{X \notin cmd_k \wedge X \in cmd_{k+1}} Ouv(X)$$

- Il faut rajouter à cela le coût de l'ouverture initiale des buses nécessaires pour le premier rang, que l'on nommera co_{init} et le coût de la fermeture finale des buses utilisées pour le dernier, que l'on nommera co_{fin} :

$$\begin{aligned} co_{init} &= Ouv(cmd_1) = \sum_{X \in cmd_1} Ouv(X) \\ co_{fin} &= Ferm(cmd_m) = \sum_{X \in cmd_m} Ferm(X) \end{aligned}$$

- SO la Séquence Optimale globale à appliquer sur le rang :

$$SO = \bigcup_{k=1}^{m} cmd_k$$

- CO le Coût Optimal global en quantité de produit pulvérisé pour le rang étudié :

$$CO = co_{init} + \sum_{k=1}^{m} co_k + co_{fin}$$

Nous supposons maintenant que ce rang est découpé en n parties $P1$, ..., Pn. Nous déclinons ces notations au niveau des parties. Soient :

- l_i le nombre de blocs de la partie Pi.
- $cmd_{i,k}$ la commande choisie pour le bloc k de la partie Pi.
- $co_{i,k}$ le coût correspondant à $cmd_{i,k}$ (y compris pour les coûts initiaux et finaux)
- SO_i la Séquence Optimale à appliquer sur la partie Pi :

$$SO_i = \bigcup_{k=1}^{l_i} cmd_{i,k}$$

- CO_i le Coût Optimal en quantité de produit pulvérisé sur la partie Pi :

$$CO_i = co_{i,init} + \sum_{k=1}^{l_i} co_{i,k} + co_{i,fin}$$

8.1.3.2 Étude d'un bloc trou

Pour un bloc de type "trou", toutes les buses vont se fermer, ce qui idéalement mène à une consommation nulle de produit. Cependant, au début d'un trou, il y aura un temps nécessaire à la fermeture des buses ouvertes pour le bloc précédent pendant lequel du produit sera consommé ; de même, à la fin d'un trou, il y aura un temps nécessaire à l'ouverture des buses utilisées pour la commande du bloc suivant (cf. les figures 4.5 et 8.3). Par contre, ces ouvertures/fermetures ne dépendent plus que de la commande du bloc suivant/précédent, celle du bloc trou étant connue. On a donc, si le bloc k est un trou :

$$cmd_k = \text{'–'} \text{ donc } \; b_k(cmd_k) = 0 \; \text{ et } \; co_k = Ferm(cmd_{k-1}) + Ouv(cmd_{k+1}) \quad (8.1)$$

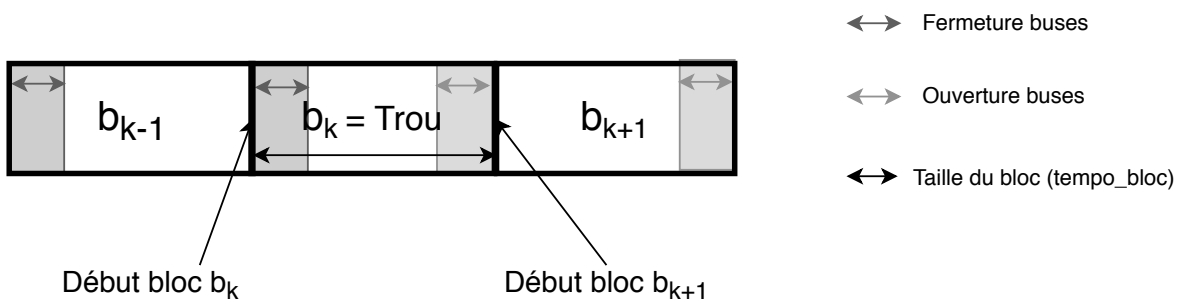


FIGURE 8.3 – Exemple d'un bloc trou

8.1.3.3 Preuve pour la décomposition

Nous allons illustrer notre preuve sur un rang composé de 5 blocs, que nous nommerons b_1, b_2, b_3, b_4 et b_5, le bloc b_3 étant un trou. Si on suppose que la durée de b_3 est supérieure à la somme des temps d'ouverture et fermeture des buses, ce qui est une hypothèse initiale de notre étape 2 AMPS, il est donc possible de faire une décomposition du rang en 2 parties : $P1$ qui regroupe les trois premiers blocs, et $P2$ qui inclus les blocs b_3 jusqu'à b_5. Pour prouver que notre méthode de décomposition garantie la conservation de la propriété de vérification et d'optimisation de pulvérisation (Propriété 3), nous allons comparer et analyser les résultats de vérification entre le rang initial et le rang décomposé.

On veut prouver que la décomposition conserve le coût optimal, c'est à dire que :

$$CO = \sum_{j=1}^{n} CO_j \tag{8.2}$$

On a pour le rang initial non décomposé :

$$SO = cmd_1 \cup cmd_2 \cup cmd_3 \cup cmd_4 \cup cmd_5 \tag{8.3}$$

On a donc, d'après l'équation 8.1 et b_3 étant un trou :

$$\begin{aligned} CO &= co_{init} + co_1 + co_2 + co_3 + co_4 + co_5 + co_{fin} \\ &= Ouv(cmd_1) + co_1 + co_2 + Ferm(cmd_2) \\ &\quad + Ouv(cmd_4) + co_4 + co_5 + Ferm(cmd_5) \end{aligned} \tag{8.4}$$

Nous allons maintenant nous intéresser à la séquence décomposée. Une question qui se pose à ce stade est la considération de la quantité de produit dans le trou. La décomposition de modélisation que nous avons effectuée est faite de sorte que dans la décomposition, le temps de fermeture des buses est considéré dans la partie antérieure au trou, alors que le temps d'ouverture des buses est considéré dans la partie postérieure. Notre décomposition permet de ne pas considérer ces quantités

de produit en double. Cela revient en fait à considérer que ce bloc n'existe pas (fermeture de toutes les buses ouvertes à la fin d'une partie, et ouverture avant le début du premier rang d'une partie de toutes les buses nécessaires pour sa commande).

On a donc pour la partie $P1$:

$$\begin{aligned} CO_1 &= co_{1,init} + co_{1,1} + co_{1,2} + co_{1,fin} \\ CO_1 &= Ouv(cmd_{1,1}) + co_{1,1} + co_{1,2} + Ferm(cmd_{1,2}) \end{aligned} \tag{8.5}$$

On a donc pour la partie $P2$ (en conservant la numérotation des blocs de la partie initiale) :

$$\begin{aligned} CO_2 &= co_{2,init} + co_{2,4} + co_{2,5} + co_{2,fin} \\ CO_2 &= Ouv(cmd_{2,4}) + co_{2,4} + co_{2,5} + Ferm(cmd_{2,5}) \end{aligned} \tag{8.6}$$

D'après les équations 8.4, 8.5, 8.6, on a donc bien le respect de notre équation 8.2, à condition que les séquences optimales soient les mêmes sans et avec décomposition.

Or on a vu dans la section 4.3.1 que le choix entre C_{best} et C_{alt} pouvait être influencé par la commande des blocs postérieurs, si l'on voit un intérêt à garder certaines buses ouvertes plutôt que de les fermer et de les rouvrir immédiatement après. Lors de la présence d'un trou, ce problème ne se pose pas : le trou oblige la fermeture de toutes les buses. Ainsi, les blocs suivant le trou n'influencent plus les commandes des blocs antérieurs au trou, ce qui rend l'optimisation des commandes avant et après le trou indépendante. Dans notre cas exemple, b_3 étant un trou, on aura donc la séquence optimale de commande des blocs b_1, b_2 qui est la même dans le rang complet que dans la partie 1, et de même pour la partie 2 et les blocs b_4 et b_5. De façon formelle, cela signifie qu'on a : $cmd_k = cmd_{i,k}$ et donc $co_k = co_{i,k}$, $\forall i \leq n$, $\forall k \leq m$.

Ce raisonnement sur un exemple générique constitue notre preuve de la décomposition par trou.

La preuve pour la conservation des propriétés de vérification et d'optimisation pour la décomposition par injection est basée sur le même principe que précédemment. La notion de neutralité vis-à-vis de la commande globale s'applique également lorsque les blocs de recouvrement sont des blocs à commande constante ($C_{best} = C_{alt}$). En effet, l'influence de la commande des blocs précédent et suivant sur la commande du bloc en cours ne peut être effective que s'il est possible de choisir une commande (donc entre C_{best} et C_{alt}) pour le bloc en cours (comme dans la figure 4.5 ou la commande du bloc 2 est modifiée). Mais cela n'est pas possible s'il n'existe qu'une seule commande possible. Nous pouvons donc appliquer la même logique de raisonnement que pour les trous, et démontrer que la propriété globale du modèle initial que nous recherchons peut être obtenue par composition des propriétés obtenues localement au niveau de chaque sous-modèle (partie) identifié.

8.2 Application de la méthodologie de décomposition au rang d'étude

8.2.1 Décomposition de la modélisation

Le rang de vigne étudié dans le chapitre 4 contient 262 blocs dont 10 de type *"Trou"*, dont les caractéristiques sont données dans la table 8.1. On remarque que le premier bloc de la vigne est un bloc de type *"Trou"* et que la taille minimale d'un bloc de type *"Trou"*, exprimée en durée, est $0.4s$.

TABLE 8.1 – Position et taille des trous dans le rang d'étude

Indice trous	Durée (s)
0	0.6
6	1.0
30	0.6
48	0.9
69	0.4
84	0.9
91	0.5
105	1.1
140	0.7
160	0.4

Ainsi, la décomposition de la modélisation permet d'avoir 10 parties qui sont décrites par la table 8.2. La colonne "Limite" permet de donner l'indice du premier et du dernier bloc formant la partie et la colonne "Taille" donne le nombre de blocs formant la partie.

On remarque dans la table 8.2 qu'il y a des parties de petite taille (7 blocs) et des parties de taille beaucoup plus grande (102 blocs). Des trop petites parties risquent de compliquer inutilement le processus de validation en décomposant trop, et inversement les trop grosses parties risquent d'exploser. L'idée consiste à assembler les parties de façon à avoir des parties de taille proche et de complexité "raisonnable" (entre 30 et 40 blocs)[2]. Il s'agit donc d'effectuer deux types d'actions : le regroupement des parties trop petites, et la décomposition (qui sera cette fois par injection) des parties trop importantes. Les nouvelles parties ainsi obtenues, appelées Qi, sont décrites par la table 8.3. Nous avons effectués les regroupement suivants[3] :

2. Cette taille sera déterminée par expertise et vérifiée empiriquement.

3. La différence de taille (1 bloc) entre $Q1$ et $P1 + P2$ vient du fait que le bloc de recouvrement (ici le trou $b6$) est compté dans chacune des sous-parties $P1$ et $P2$ et une seule fois dans $Q1$

TABLE 8.2 – Description des parties du rang d'étude

Partie	Limite	Taille
P1	b0-b6	7
P2	b6-b30	25
P3	b30-b48	19
P4	b48-b69	22
P5	b69-b84	16
P6	b84-b91	8
P7	b91-b105	15
P8	b105-b140	36
P9	b140-b160	21
P10	b160-b261	102

$Q1 = P1 \cup P2$; $Q2 = P3 \cup P4$; $Q3 = P5 \cup P6 \cup P7$. Les parties $P8$ et $P9$ restent identiques, devenant respectivement $Q4$ et $Q5$.

La partie $P10$ par contre est trop grande pour pouvoir être traitée par model-checking et elle doit donc être décomposée. Nous allons procéder à une décomposition par injection. Entre le bloc d'indice 160 et le bloc d'indice 261, il y a 49 blocs où $C_{best} = C_{alt}$. Ces blocs ont les indices [160, 161, 163, 164, 165, 167, 168, 170, 171, 174, 179, 180, 184, 185, 187, 188, 189, 191, 193, 194, 196, 198, 201, 202, 204, 205, 206, 207, 209, 210, 214, 215, 224, 226, 227, 229, 235, 238, 239, 242, 243, 244, 246, 247, 252, 253, 255, 257, 259]. Plusieurs décompositions sont possibles, nous avons choisi de décomposer $P10$ en 3 sous-parties $Q6$, $Q7$ et $Q8$ qui correspondent à la taille moyenne choisie.

TABLE 8.3 – Description des nouvelles parties

Partie	Limite	Taille
Q1	b0-b30	31
Q2	b30-b69	40
Q3	b69-b105	37
Q4	b105-b140	36
Q5	b140-b160	21
Q6	b160-b194	35
Q7	b194-b224	31
Q8	b224-b261	38

L'application de nos techniques de décomposition nous a ainsi permis d'obtenir 8 parties globalement équilibrées en taille permettant de réduire significativement la complexité de la vérification pour chacune d'elles.

Après la formation des parties formant le rang, on génère les modèles $SMQi$ (fichiers .xml) relatif à chacune des parties Qi et on génère deux fichiers .xml spécifiques (modèles + informations d'injection) pour la vérification par injection, le premier regroupant les 2 parties $Q6$ et $Q7$ ($SMQ7^{6}$) et le deuxième regroupant les 3 parties $Q6$, $Q7$ et $Q8$ ($SMQ8^{6-7}$) .

8.2.2 Décomposition de la vérification

La décomposition de la vérification consiste à vérifier la propriété 3 sur les sous-modèles des parties du rang. Les résultats sont décrits dans la table 8.4 pour chacune des parties Qi identifiées. Dans ce tableau, la colonne "Limite" est un peu différente des tableaux précédents pour les parties où on applique la décomposition par injection : en italique est indiqué le bloc initial du sous-modèle, à partir duquel on a injecté leur commande optimale, alors qu'en écriture normale ce sont les blocs ayant la liberté de commande. La colonne nommée "Coût" donne le coût optimal CO_i en quantité de produit pulvérisé pour la partie Qi. Pour les lignes concernant la vérification de parties avec injection, cette colonne donne par définition le coût total des blocs indiqués. La colonne nommée "TE" (Temps d'Exécution) donne le temps total requis pour la vérification de la propriété et la colonne nommée "M" (Mémoire) donne la mémoire requise pour la vérification de la propriété.

Nous pouvons observer dans ce tableau 8.4 que la vérification de la propriété 3 sur les parties $Q4$ et $Q8$ a produit un problème d'explosion combinatoire. Nous avons dû de nouveau appliquer la méthode de décomposition par injection sur ces deux parties, ce qui a mené à la création de 2 parties supplémentaires.

On peut également remarquer que la vérification de la propriété 3 sur les parties $Q2$ et $Q3$, elles aussi de tailles importantes, n'a pas conduit à une explosion combinatoire. Ceci se justifie par le fait que la complexité de la vérification n'est pas uniquement dépendante du nombre de blocs. Elle est aussi dépendante du nombre de choix par blocs, c'est-à-dire ici du nombre de blocs où $C_{best} = C_{alt}$. Par exemple, dans la partie $Q2$, pourtant de taille supérieure (40 blocs), il y a 25 blocs où $C_{best} = C_{alt}$ ce qui limite fortement la complexité, tandis que dans la partie $Q4$ il n'y a que 15 blocs où $C_{best} = C_{alt}$.

Avec ces décompositions, nous avons donc pu obtenir le coût optimal global CO en quantité du produit pulvérisé pour le rang étudié, qui est égal à la somme des coûts de chaque partie décomposée par trou, ainsi que le coût global du sous-modèle complet de la partie décomposée par injection. On a donc :

$$CO = CO_1 + CO_2 + CO_3 + CO_{4\_2} + CO_5 + CO_6 + CO_7 + CO_{8\_2} = 4668$$

Ce coût optimal correspond à une commande de pulvérisation qui elle-même est optimale, et que nous allons étudier ci-dessous.

TABLE 8.4 – Coût optimal des parties formant le rang d'étude

Partie	Limite	Coût (CO_i)	TE (s)	M (KB)
Q1	b0-b30	460	7.81	169 972
Q2	b30-b69	748	4.50	120 200
Q3	b69-b105	596	36.34	912 948
Q4_1	b105-b137	*492*	57.56	1 158 136
Q4_2	*b105*-b137-b140	548	0.10	76
Q5	b140-b160	398	0.90	16 736
Q6	b160-b194	*658*	6.51	155 928
Q7	*b160*-b194-b224	*1184*	14.12	419 416
Q8_1	*b160*-b224-b252	*1722*	12.41	432 416
Q8_2	*b160*--b252-b261	1918	0.10	76
Rang complet	b160-b261	4668	140,35	1 158 136

8.2.3 Séquence de commande optimale obtenue

La Figure 8.4 montre la séquence de contrôle optimale avec le coût le plus bas obtenue sur notre exemple. Sa légende rappelle qu'à chaque type de commande, une couleur est affectée. Dans cette séquence, CH est la commande la plus fréquente. Elle apparaît dans 94 blocs de végétation. Ensuite, on trouve la commande LH & CH (magenta), elle apparaît dans 83 blocs de végétation. Au contraire, la commande CH & HH (jaune) est présente seulement 6 fois. Il apparaît dans la solution optimale qu'une main seule est suffisante pour pulvériser 42% des blocs, alors qu'un pulvérisateur classique non contrôlé dynamiquement utiliserait toujours deux mains. Souvent, en pulvérisation de précision, le temps de réponse des buses n'est pas pris en compte, alors que notre méthode AMPS prend en considération cet aspect.

Pour montrer l'apport de notre méthode, nous allons comparer la configuration de la solution optimale, que nous avons obtenue grâce à notre méthode, à trois autres configurations de commande : la pulvérisation classique qui utilise toujours 2 mains, la pulvérisation théoriquement idéale en appliquant seulement les commandes à C_{best} et sans prendre en compte le temps de réponse des buses, et enfin la pulvérisation en appliquant seulement les commandes à C_{best} et en prenant en compte le temps de réponse des buses. Le tableau 8.5 est un tableau comparatif des économies de produit réalisées par rapport à la pulvérisation classique.

La configuration *Pulvérisateur classique* consiste à utiliser les mains $LH\&HH$ pour tout le rang, ce qui résulterait pour les 262 blocs à une pulvérisation en débit nominal pendant 137,2 secondes. Si nous disposons maintenant de la possibilité de commander séparément et dynamiquement les buses, nous avons d'abord considéré la meilleure séquence de contrôle théorique, en appliquant une cartographie C_{best} sans tenir compte de la dynamique du pulvérisateur (c'est-à-dire en supposant un temps d'ouverture et de fermeture instantanés). Cette séquence permettrait, s'il elle

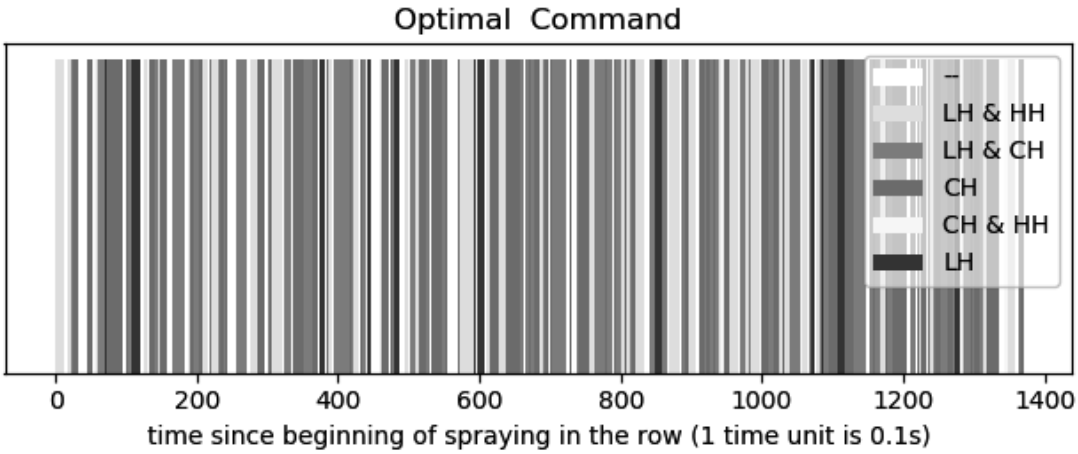


FIGURE 8.4 – La commande optimale calculée

TABLE 8.5 – Tableau comparatif de configurations de pulvérisation

Configurations	coût	économie
(1) Pulvérisateur classique	5488	0 %
(2) ~~commandes C_{best} - dynamique du pulvérisateur~~	4188	~~23,69 %~~
(3) **Séquence optimale** (AMPS)	4668	**14,94** %
(4) commandes C_{best} + dynamique du pulvérisateur	4916	10,42 %

était possible, d'économiser 23,69% par rapport à la quantité de pulvérisation classique. Mais la dynamique du pulvérisateur est une contrainte réelle, et doit être considérée. Or, les résultats réels de l'application des commandes C_{best} pour chacun des blocs ne permettent finalement d'économiser que 10,42% du produit. Pour terminer, l'application de la séquence optimale (un mélange des commandes C_{best} et C_{alt}) obtenue grâce à notre méthode permet d'économiser 14,94% du produit tout en maintenant une protection locale suffisante. Nous avons donc ici une preuve expérimentale de l'efficacité de notre méthode.

La méthode AMPS réduit donc significativement les quantités de produits chimiques pulvérisés tout en maintenant le rendement et la qualité. Elle prend en compte la dynamique de contrôle d'un pulvérisateur. Ce résultat est plus réaliste qu'une carte statique avec un point de consigne unique pour chaque bloc, et plus efficace que les méthodes classiques généralement employées.

8.3 Conclusion

Nous avons décrit dans ce chapitre la méthode de décomposition déclinée spécifiquement sur un problème issu de l'agriculture de précision de dimension spatiale 1D : la pulvérisation d'un rang de vigne. Cette méthode est basée sur le principe générique de décomposition, décrit dans le chapitre 7. La décomposition en modélisation consiste à décomposer le rang complet en n parties de façon à avoir entre elles un bloc de recouvrement de type "Trou". Ce type de décomposition est appelé "décomposition orientée trou". La décomposition en vérification consiste à vérifier la propriété 3 sur chaque partie du rang ce qui permet de calculer la séquence et les coûts optimaux pour chaque partie. L'analyse de la décomposition consiste à prouver que l'union des séquences optimales des parties permet d'obtenir la séquence optimale complète et que la somme des coûts optimaux des parties permet d'obtenir le coût optimal global.

Le processus de vérification de la propriété 3 pour les parties formées entre blocs de type "Trou" peut engendrer un problème d'explosion combinatoire lorsque ces parties sont de trop grandes dimensions. Nous avons donc proposé une approche complémentaire dite approche de décomposition par injection.

Nous avons appliqué cette méthode à un exemple réel et nous avons montré que la méthode AMPS est maintenant applicable à des données LiDAR issues d'une vigne réelle composée de rangs de grandes dimensions.

La séquence de contrôle optimale ainsi obtenue indique que la réduction des produits phytosanitaires est possible tout en maintenant une protection suffisante sur chaque plant de vigne. La méthode AMPS prend en compte la dynamique de contrôle du pulvérisateur utilisé. Si la dynamique du pulvérisateur n'était pas prise en compte, les économies potentielles sur les produits phytosanitaires seraient surestimées. Le principal intérêt de la méthode AMPS est d'estimer, avant le développement d'un système spécifique, les gains qui pourraient être réalisés avec un pulvérisateur automatisé.

La limite principale de notre méthode de décomposition est qu'elle n'est pas applicable si dans un rang, il n'y a pas de trous ou il n'y a pas assez de blocs où $C_{best} = C_{alt}$. L'exemple réel étudié nous a cependant montré que la réalité a plutôt tendance a favorisé les "états stables" propice à la décomposition. Dans le cas contraire, nous pouvons agir sur l'étape 2 de l'AMPS (découpage du rang en blocs en fonction des données LIDAR), et en particulier sur nos dimensions d'échantillonage et sur nos règles d'agrégation pour générer en priorité des blocs avec ce profil.

Une perspective de nos travaux serait d'aller vers un outil utilisable pour la communauté agricole. La méthode AMPS pourrait par exemple être automatisée et intégrée dans un outil graphique. Cet outil prendrait en entrée les données LiDAR des parcelles de vigne et les caractéristiques d'un pulvérisateur et fournirait en sortie la séquence optimale de commandes de configuration de pulvérisation à appliquer

dans le rang. Ceci pourrait servir d'indication à l'humain en charge du pulvérisateur. En allant plus loin, on pourrait intégrer cet outil au système de commande d'un robot pulvérisateur autonome, afin de générer son contrôleur de pulvérisation.

Une autre perspective est d'appliquer la méthode dans un ensemble de parcelles avec des caractéristiques de végétation très différentes d'une parcelle à l'autre et d'analyser le potentiel de l'automatisation pour réduire les quantités pulvérisées dans différents vignobles.

9 Décomposition et question d'atteignabilité sur une grille 2D

Sommaire

9.1 Rappel de la contribution de grille 2D

Dans le chapitre 6, nous avons étudié le problème d'un robot agissant localement selon une grille spatiale. Dans chaque case de la grille, le robot effectue une seule action. Le robot est soumis à plusieurs contraintes : des contraintes de déplacement, des contraintes de visite et des contraintes de précédence entre les actions. Pour modéliser ce type de comportement, nous avons proposé le Modèle Région de Déplacement (MRD) qui permet de représenter cette grille spatial 2D. Le problème de vérification étudié est celui de l'atteignabilité d'un bord de la grille à l'autre. Les premiers résultats montrent qu'un problème d'explosion combinatoire se produit au moment de la vérification lorsque la grille a une taille significative.

Dans ce chapitre, nous proposons une méthodologie de décomposition réduisant les besoins en mémoire du processus de vérification. Nous nous appuyons sur le principe générique de décomposition décrit dans le chapitre 7, qui consiste à décomposer le modèle, décomposer la vérification et analyser cette décomposition pour prouver la justesse de la vérification.

Le plan de ce chapitre s'énonce comme suit : la section 2 présente une description générale de la méthodologie de décomposition appliquée au problème d'atteignabilité sur une grille 2D. La section 3 résume les concepts retenus et présente les principales phases de l'algorithme de décomposition proposé. Dans la section 4, des exemples d'exécution expliquant le fonctionnement de l'algorithme de décomposition sont décrits. Enfin dans la section 5, une étude comparative entre l'algorithme de décomposition et l'algorithme de Référence est fournie.

9.2 Description générale de la méthodologie de décomposition

La méthodologie de décomposition générique doit être adaptée à chacun de nos problèmes. Nous décrivons ici sa déclinaison sur un problème de grille 2D.

9.2.1 Décomposition de la modélisation

Dans le cas de la vérification de la propriété d'atteignabilité bord à bord sur grille 2D, la décomposition de la modélisation que nous proposons consiste à décomposer la grille globale G en deux sous-grilles (S_1 et S_2). Les sous-grilles sont définies de telle façon à avoir entre elles une colonne de recouvrement (RC) qui appartient à ces deux sous-grilles. On rappelle que, pour une grille G, B_G représente l'ensemble des cases du bord gauche de la grille et B_D l'ensemble des cases du bord droit. Formellement, la colonne de recouvrement est notée :

$$\mathcal{RC}(\mathcal{G}) = \mathcal{B}_{\mathcal{D}}(S_1) = \mathcal{B}_{\mathcal{G}}(S_2)$$

Par exemple, pour la grille 4 lignes × 8 colonnes de la figure 9.1, les 5 premières colonnes forment la sous-grille S_1, les 4 dernières colonnes forment la sous-grille S_2. La colonne n°5 est la colonne de recouvrement RC, qui appartient donc aux 2 sous-grilles. Nous appellerons cette technique de décomposition *la décomposition avec recouvrement*.

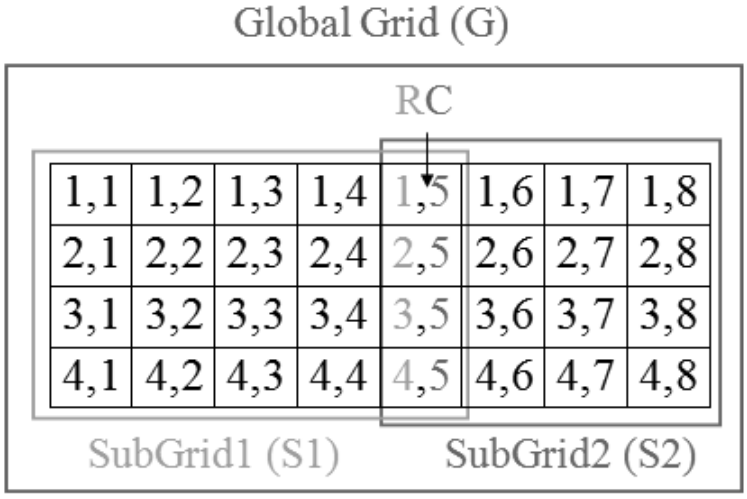


FIGURE 9.1 – Décomposition d'une grille en deux sous-grilles

9.2.2 Décomposition de la vérification

Rappelons la propriété d'atteignabilité totale bord à bord (Π) utilisée dans le chapitre 6 :

$$\forall\ g \in\ \mathcal{B}_{\mathcal{G}}(G), \forall\ d \in\ \mathcal{B}_{\mathcal{D}}(G), \exists\ g \longrightarrow d\ \ (\Pi)$$

L'idée pour la décomposition en vérification consiste à trouver des points particuliers qui permettent le passage de la sous-grille S_1 à la sous-grille S_2. En d'autres termes, il s'agit de trouver des points de la colonne de recouvrement qui sont atteignables depuis le bord de gauche de S_1 et permettent d'atteindre le bord de droite de S_2. Nous définissons deux types de décomposition de vérification : la *décomposition par points critiques* et la *décomposition par répartition*. Pour expliquer ces deux types de décomposition, nous avons besoin d'introduire quelques terminologies :

- On appelle *point potentiel* un point de la colonne de recouvrement atteignable à partir de tous les points du bord de gauche de S_1 ($\mathcal{B}_{\mathcal{G}}(S_1)$).
- On appelle *point critique* un point potentiel permettant également d'atteindre tous les points du bord de droite de S_2 ($\mathcal{B}_{\mathcal{D}}(S_2)$).

Dans la suite la décomposition par point critique est expliqué.

9.2.2.1 Décomposition par points critiques

Soient les deux ensembles P_1 et P_2 suivants :

$$\begin{aligned} P_1 &= \{u \in \mathcal{B}_{\mathcal{D}}(S_1) \ / \ \forall\, g \in \mathcal{B}_{\mathcal{G}}(S_1), g \longrightarrow u\} \\ P_2 &= \{v \in \mathcal{B}_{\mathcal{G}}(S_2) \ / \ \forall\, d \in \mathcal{B}_{\mathcal{D}}(S_2), v \longrightarrow d\} \end{aligned}$$

avec $g \longrightarrow u$ la fonction permettant de garantir qu'il existe un chemin entre un point g et un point u d'une grille. Ce chemin doit respecter les contraintes de déplacement, les contraintes de précédence entre les actions et les contraintes de visite de notre problème. Ainsi, P_1 (resp. P_2) représente l'ensemble des points de la colonne de recouvrement reliés par un chemin à tous les points du bord gauche de S_1 (resp. bord droit de S_2).

La décomposition en vérification consiste à décomposer la propriété (Π) en deux propriétés Π_1 et Π_2, qui représentent les propriétés d'atteignabilité d'un bord à l'autre pour chacune des sous-grilles. Π_1 (resp. Π_2) est vérifiée s'il existe au moins un élément dans P_1 (resp. P_2) :

$$\begin{aligned} P_1 &\neq \emptyset \quad (\Pi_1) \\ P_2 &\neq \emptyset \quad (\Pi_2) \end{aligned}$$

Nous prouvons dans la section suivante que si la sous-grille S_1 vérifie P_1 et que S_2 vérifie P_2, et s'il existe un et un seul point appartenant aux deux ensembles P_1 et P_2, alors la propriété (Π) est vérifiée sur G. Ce point appartenant aux deux ensembles est un *point critique* et il vérifie à la fois les propriétés Π_1 et Π_2.

$$P_1 \cap P_2 \neq \emptyset \text{ sous certaines conditions} \quad \Longrightarrow \ \Pi \text{ est vérifiée sur G.}$$

Dans la suite, la décomposition par répartition est décrite.

9.2.2.2 Décomposition par répartition

Le cas précédent est un cas particulier, basé sur l'hypothèse que tous les chemins passent par un point unique de la colonne de recouvrement, ce qui ne représente pas la majorité des cas. Nous avons donc proposé une seconde méthode de décomposition plus générique : la décomposition par répartition. L'idée est de trouver un ensemble de points appartenant à la colonne de recouvrement qui, lorsqu'on assemble tous les chemins qui passent par ces points, permettent de remplir la propriété Π.

La décomposition en vérification ici se base sur des propriétés proches de Π_1 et Π_2, mais en considérant les chemins de façon indépendante.

Soient les deux ensembles P_3 et P_4 suivants :

$$\begin{aligned} P_3 &= \{g \in \mathcal{B}_{\mathcal{D}}(S_1) \ / \ \exists u \in \mathcal{B}_{\mathcal{G}}(S_1), u \longrightarrow g\} \\ P_4 &= \{g \in \mathcal{B}_{\mathcal{G}}(S_2) \ / \ \exists v \in \mathcal{B}_{\mathcal{D}}(S_2), g \longrightarrow v\} \end{aligned}$$

P_3 (resp. P_4) représente l'ensemble des points de la colonne de recouvrement tels qu'il existe au moins un point du bord gauche de S_1 (resp. bord droit de S_2) avec lequel il est relié par un chemin.

On peut donc se baser sur deux nouvelles propriétés pour la décomposition :

$$\begin{aligned} P_3 &\neq \emptyset \quad (\Pi_3) \\ P_4 &\neq \emptyset \quad (\Pi_4) \end{aligned}$$

L'intersection de ces deux ensembles $P_3 \cap P_4$ représente l'ensemble des points de la colonne de recouvrement reliés à la fois à au moins un point du bord gauche et au moins un point du bord droit de la grille G. Et sous condition de respecter les contraintes de notre problème, en particulier les contraintes de visite, alors cela représente un ensemble de chemins reliant chacun un point du bord de gauche de la grille à un point du bord droit. Alors, si l'union de tous ces chemins permet de relier tous les points du bord de gauche et tous les points du bord de droite, Π est vérifiée sur G.

9.2.3 Analyse de la décomposition

9.2.3.1 Preuve de la décomposition par point critique

Supposons que la propriété (Π_1) soit vérifiée sur S_1 et que la propriété (Π_2) est vérifiée sur S_2. Ceci représente l'ensemble (P) des conditions suivantes :

$$\left\{ \begin{array}{l} S1 \cap S2 = RC \\ S1 \cup S2 = G \\ (\Pi_1)\ vérifiée\ dans\ S_1 \\ (\Pi_2)\ vérifiée\ dans\ S_2 \end{array} \right. \quad (P)$$

Considérons e_1 un point de la colonne de recouvrement avec $e_1 \in P_1$. Alors, pour tout point de $\mathcal{B}_{\mathcal{G}}(S_1)$, il existe un chemin qui part de ce point et arrive en e_1. Par définition, ce point est un *point potentiel* : il peut potentiellement être un point critique. De même, si $e_2 \in P_2$ alors il existe des chemins qui partent du point e_2 et qui arrivent à tous les points de $\mathcal{B}_{\mathcal{D}}(S_2)$. Il suffit alors que $e_1 = e_2$ pour que nous ayons identifié un *point critique*.

Il s'agit cependant de vérifier que les contraintes de notre système restent vérifiées dans la colonne de recouvrement lorsque l'on compose les chemins de la sous-partie gauche avec ceux de la sous-partie droite. Toutes ces contraintes sont vérifiées de fait par la vérification des propriétés locales Π_1 et Π_2 : s'il existe un chemin entre deux points, alors les contraintes sont forcément respectées sur ce chemin. Seule la colonne de recouvrement doit être analysée en détail pour prouver notre décomposition. En particulier, la contrainte de visite unique impose qu'une case de la colonne de recouvrement ne peut être parcourue qu'une seule fois. Les autres contraintes seront

alors vérifiées de fait. Ainsi, pour un point potentiel donné, pour que l'assemblage d'un chemin appartenant à la sous-partie de gauche avec un chemin appartenant à la sous-partie de droite respecte les contraintes de notre système, il faut que l'intersection des ensembles de ces chemins soit réduite à un unique point qui est le point potentiel lui-même (cf. Figure 9.2).

Soit e un point potentiel : $e \in P1$. $C_1(e)$ est l'ensemble des points de la colonne de recouvrement devant être visités pour que la propriété (Π_1) soit vérifiée pour le point e. Par exemple sur la Figure 9.2, l'ensemble de point de RC devant être parcourus pour atteindre $e1$ à partir de tous les points de gauche est $C_1(e_1) = \{x_1, e_1\}$. De même, soit $C_2(e)$ l'ensemble des points de la colonne de recouvrement devant être visités pour que la propriété (Π_2) soit vérifiée pour $e \in P2$. Sur la figure 9.2, l'ensemble de point de RC devant être parcourus pour pouvoir atteindre tous les points de droite à partir de e_1 est $C_2(e_2) = \{e_1, x_2, x_3\}$.

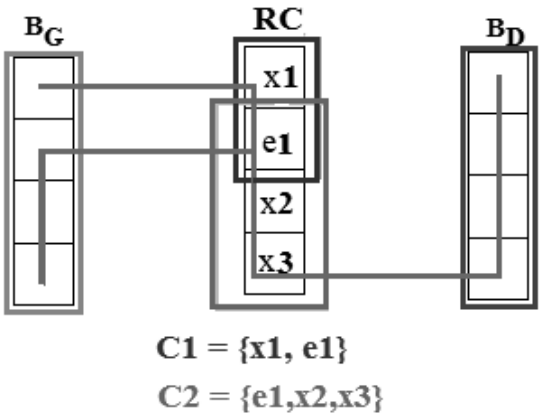


FIGURE 9.2 – Schéma illustratif pour la preuve de décomposition par point critique

Pour qu'il existe un point critique respectant les contraintes du système, et donc que (Π) soit vérifiée sur G, les conditions suivantes doivent donc être respectées :

$$\begin{cases} P_1 \cap P_2 \neq \emptyset \\ \exists e \in RC \mid e \in P_1 \cap P_2 \text{ et } C_1(e) \cap C_2(e) = \{e\} \end{cases} \quad (Q)$$

Ces conditions garantissent qu'ils existent bien des chemins partant de tous les points du bord gauche vers tous les points du bord droit, qui passent tous par le point critique e, et que les contraintes du système sont respectées.

Sur notre exemple de la figure 9.2, nous avons bien les conditions de (Q) vérifiées, avec : $C_1 \cap C_2 = \{e_1\}$. Par conséquence, la propriété (Π) est vérifiée sur G avec le point critique e_1.

9.2.3.2 Preuve de la décomposition par répartition

De même que pour la preuve précédente, supposons que la propriété (Π_3) est vérifiée sur S_1 et que la propriété (Π_4) est vérifiée sur S_2. Ceci représente l'ensemble (R) des conditions suivantes :

$$\begin{cases} S1 \cap S2 = RC \\ S1 \cup S2 = G \\ (\Pi_3)\ vérifiée\ dans\ S_1 \\ (\Pi_4)\ vérifiée\ dans\ S_2 \end{cases} (R)$$

Nous allons dans un premier temps reconstruire à partir de P_3 et P_4 l'ensemble des chemins reliant des points du bord de gauche à des points du bord de droite et respectant les contraintes du système dans G. Pour tout couple (e, f) tel que $e \in \mathcal{B}_{\mathcal{G}}(G)$ et $f \in \mathcal{B}_{\mathcal{D}}(G)$, il existe un chemin complet du bord gauche de la grille jusqu'à son bord droit s'il existe un chemin entre les bords droit et gauche des deux demi-grilles, et si ces chemins ont un et un seul point commun au niveau de la colonne de recouvrement.

Soient $C_3(e, g, f)$ (resp. $C_4(e, g, f)$) l'ensemble de points de la colonne de recouvrement devant être parcourus dans P_3 (resp. P_4) pour établir le chemin de $e \in \mathcal{B}_{\mathcal{G}}(G)$ à $f \in \mathcal{B}_{\mathcal{D}}(G)$ passant par $g \in P_3 \cap P_4$. Si $C_3(e, g, f) \cap C_4(e, g, f) = \{g\}$ alors les contraintes de visite sont vérifiées et on a $e \longrightarrow f$ dans G.

La propriété Π est vérifiée sur G si pour chaque couple de points gauche/droite, il existe un chemin complet, c'est-à-dire si :

$$\forall (e, f) \in \mathcal{B}_{\mathcal{G}}(G) \times \mathcal{B}_{\mathcal{D}}(G), \quad \exists g \in RC \mid C_3(e, g, f) \cap C_4(e, g, f) = \{g\}$$

Dans la section suivante, l'algorithme de décomposition est détaillé.

9.3 Algorithme de décomposition

L'algorithme décrit dans cette section concerne la décomposition de la modélisation et la décomposition de la vérification d'une grille 2D. Notre objectif est d'évaluer notre approche de décomposition sur des grilles de différentes dimensions et pour des configurations d'actions choisies aléatoirement.

Nous utilisons un organigramme pour décrire la structure de notre algorithme de décomposition (figure 9.3). Les procédures et les traitements sont représentés par des rectangles. Les tests sont représentés par des losanges, le début et la fin sont représentés par des cercles. L'algorithme de décomposition est composé de 3 fonctions :

- La fonction *Initialisation(Grid)* : définit et initialise les structures de données et les variables utilisées.

- La fonction *AlgDécMod()* : gère la décomposition de la modélisation.
- La fonction *AlgDécVerif()* : gère la décomposition de la vérification

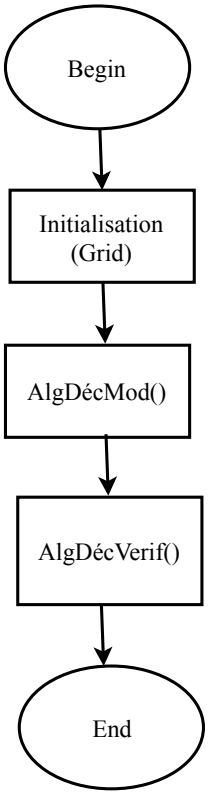


FIGURE 9.3 – Organigramme de l'algorithme de décomposition

9.3.1 Algorithme de la fonction d'*Initialisation*

La fonction *Initialisation* permet d'initialiser les structures de données et les variables nécessaire au déploiement de l'algorithme de décomposition. Elle assure aussi la génération aléatoire des actions à réaliser par le robot dans la grille principale G.

L'objectif de la génération aléatoire des actions est de générer automatiquement des grilles de dimensions données pour vérifier la propriété d'atteignabilité totale bord à bord sur ces grilles. Cependant dans beaucoup de cas, le caractère aléatoire de la génération des actions conduira à la non vérification de la propriété d'atteignabilité. C'est pourquoi, au moment de la génération des actions, il y a une partie fixe et une partie générée aléatoirement. La partie fixe impose la vérification de la propriété. Cette partie fixe est nommée *Grid* et elle est passée en paramètre à la fonction d'initialisation.

Le pseudo code de la fonction *Initialisation* est décrit en annexe A.

9.3.2 Algorithme de décomposition de la modélisation

L'algorithme de décomposition de la modélisation *AlgDécMod()* consiste à choisir une colonne de recouvrement se trouvant au centre de la grille (ou à son voisinage). Comme pour les approches de décompositions de modélisation des chapitres précédents, la colonne de recouvrement appartiendra au deux sous-grilles ainsi créées. Ensuite, l'algorithme génère le modèle de chaque sous-grille sous la forme d'un fichier xml. Ces deux fichiers générés seront utilisés par l'outil UppAal en utilisant les fonctions `memtime` [1] et `verifyta` [2].

Le pseudo code de la fonction *AlgDécMod()* est donné en annexe A.

9.3.3 Algorithme de décomposition de la vérification

L'algorithme de décomposition de la vérification proposé décompose la vérification en plusieurs phases. On commence par chercher dans la colonne de recouvrement les points potentiels (points atteignables à partir de tous les points de $\mathcal{B}_{\mathcal{G}}(G)$). Puis, on cherche à savoir si, parmi ceux-ci, certains sont des points critiques (points permettant d'atteindre tous les points du bord de droite de G). On appelle l'ordre du point potentiel e le nombre de points de la colonne de recouvrement qui doivent être visités pour atteindre e.

Les différentes phases mises en œuvre pour la décomposition de la vérification d'une grille 2D sont présentées synthétiquement dans l'organigramme de la figure 9.4. L'algorithme de décomposition (*AlgDécVerif()*) est un algorithme itératif, composé principalement de 4 fonctions : *InitDecVerif()*, *CalculPP()*, *CalculPC()* et *RepartitionG2()*. Le pseudo code de cet algorithme est décrit dans l'algorithme 3 et globalement expliqué ci-dessous. Il sera illustré sur des exemples dans la section 9.4.

- L'algorithme commence par appeler la fonction *InitDecVerif()*. Cette fonction initialise la variable *order* à 1, les variables $finish$ et $ResolvedProblem$ à `faux`. La variable *order* désigne l'ordre du point potentiel, la variable $finish$ vaut `vraie` lorsque le processus de vérification est terminée et la variable $ResolvedProblem$ désigne si la propriété Π est vérifiée ou non.

- Ensuite, tant que la valeur de la variable $finish$ est `faux` et que l'ordre *order* est inférieur à la largeur de la grille, l'algorithme *AlgDécVerif()* continue. Il appelle en premier la fonction *CalculPP()*. Cette dernière recherche les points potentiels d'ordre *order* et elle remplit la structure de données PP. PP est un tableau de liste de taille égale à $largeur$ et $PP[x]$ contient la liste des points potentiels d'ordre x+1. La recherche des points potentiels consiste à trouver les points de la colonne de recouvrement tels que chaque point en question peut être atteint depuis n'importe quel point du bord gauche à un ordre donné.

1. Pour récupérer les performances d'exécution
2. Pour vérifier les propriétés par model checking dans UppAal

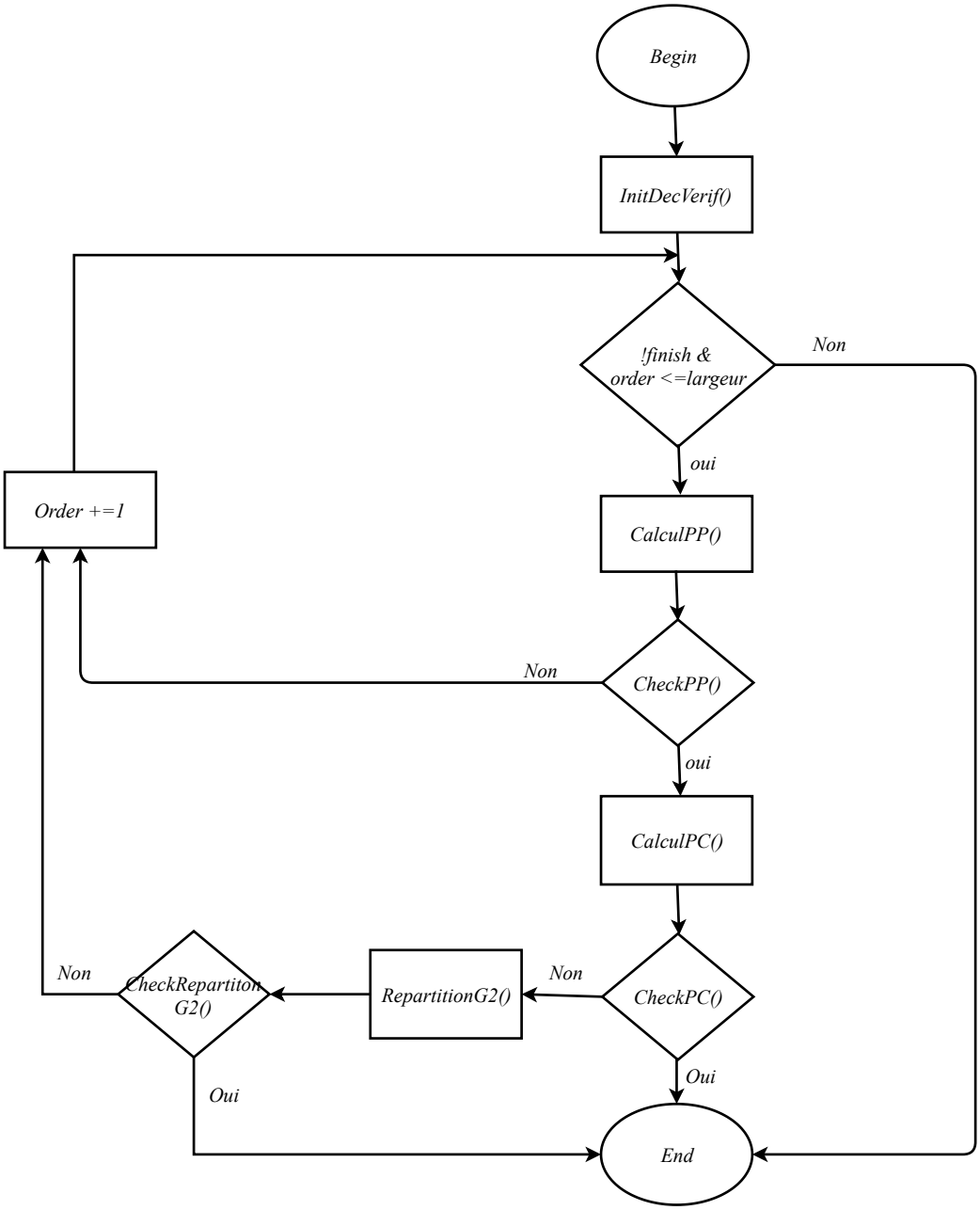


FIGURE 9.4 – Organigramme de l'algorithme de décomposition de la vérification

Elle vérifie la propriété Π_1 en effectuant des requêtes d'atteignabilité sur le sous-modèle de $S1$. La fonction *CheckPP()* retourne `vrai` s'il existe des points potentiels d'ordre *order* et retourne `faux` s'il n'y en a pas. Son pseudo code est décrit en annexe A.

Algorithm 3 AlgDécVerif()

```
1: procedure AlgDécVerif
2:     InitDecVerif() ;
3:     while finish = False and order ≤ largeur do
4:         CalculPP();
5:         if CheckPP() == False then
6:             order = order + 1;
7:         else
8:             CalculPC();
9:             if CheckPC() == True then
10:                finish = True;
11:                ResolvedProblem = True;
12:            else
13:                RepartitionG2();
14:                if CheckRepartitionG2() == True then
15:                    ResolvedProblem = True;
16:                    finish = True;
17:                else
18:                    order = order + 1;
19:                end if
20:            end if
21:        end if
22:    end while
23: end procedure
```

- Si la fonction *CheckPP()* retourne `faux` alors on répète le processus en cherchant des points potentiels d'ordre supérieur. Ainsi, la valeur de la variable *order* s'incrémente, et l'algorithme boucle de nouveau jusqu'à trouver un ordre où il existe des points potentiels. Si *order* atteint sa valeur maximale (*largeur*) sans avoir trouver de points potentiels, alors l'algorithme s'arrête : on ne peut pas vérifier la propriété en faisant une décomposition par point critique.

- Si la fonction *CheckPP()* retourne `vrai` alors la fonction *CalculPC()* est appelée. Cette dernière exploite la liste $PP[order-1]$ pour rechercher les points critiques parmi les points potentiels trouvés, et remplit la structure de données PC. PC est une liste de taille égale à *largeur* qui contient les points critiques. La recherche d'un point critique est similaire à la recherche des points potentiels : elle consiste à trouver un point de la colonne de recouvrement qui permet d'atteindre tous les points du bord de droite dans la sous-grille $S2$. Elle vérifie donc la propriété Π_2 en effectuant des requêtes d'atteignabilité sur le sous-modèle de $S2$. La fonction *CheckPC()* retourne `vrai` s'il existe des points critiques et `faux` sinon. Son pseudo code est décrit en annexe A.

- Si *CheckPC()* retourne `vrai` alors l'algorithme se termine et la propriété Π est vérifiée sur G (les variables $finish$ et $ResolvedProblem$ valent `vrai`).

- Si *CheckPC()* retourne `faux` alors il est impossible de vérifier Π avec une décomposition par point critique. On appelle alors la fonction *RepartitionG2()*. Cette dernière cherche si tous les points du bord de droite sont atteignables par les points potentiels ou non. Si c'est le cas, elle retourne `vrai` et l'algorithme se termine : la propriété Π est vérifiée. Sinon, on répète le processus de recherche des points potentiels à un ordre supérieur

Notre algorithme de décomposition est basé sur la présence de points potentiels. S'il n'existe aucun point potentiel, notre algorithme ne résout pas le problème même s'il existe une solution. En effet, la décomposition par répartition que nous avons décrit en section 9.2.2.2 est implémentée dans l'algorithme de vérification *AlgDécVerif()* (Algorithme 3) seulement sur la grille $S2$: en cas d'absence de point critique, on procède par répartition. Le même principe devrait être appliqué en cas d'absence de points potentiels sur la grille $S1$. Il suffit de chercher si tous les points du bord de gauche de la grille G peuvent être visités depuis un ensemble de points de la colonne de recouvrement. Faute de temps, la répartition sur la sous-grille $S1$ n'a pas été implémentée dans l'algorithme *AlgDécVerif()* présenté ici.

9.4 Exemples didactiques de mise en œuvre de l'algorithme de décomposition en vérification

Dans cette partie, nous expliquons le fonctionnement de l'algorithme de décomposition en vérification à travers 4 exemples.Dans le premier, le problème sera résolu à l'ordre 1 (c'est à dire avec des points potentiels d'ordre 1) et un point critique. Dans le deuxième, le problème sera résolu à l'ordre 1 mais sans point critique, donc par répartition. Dans le troisième, le problème sera résolu à l'ordre 2. Enfin, le dernier exemple montre un cas ne pouvant être résolu avec notre approche.

9.4.1 Exemple 1 : Résolution avec points potentiels d'ordre 1 et point critique

L'exemple étudié est celui de la figure 9.5. C'est une grille 4*8. Les chiffres 1, 2 et 3 au centre des cases représentent respectivement les actions A, B et C réalisées par le robot dans chaque case. L'automate de précédence des actions est le même que celui de la figure 6.3. Les nombres en petite police désignent l'indice des cases. Le rectangle vert encadre la colonne de recouvrement, avec les points d'indices [5,13,21,29]. Les lignes rouges et bleus matérialisent les chemins identifiés par l'algorithme pour relier les bords de la grille à RC.

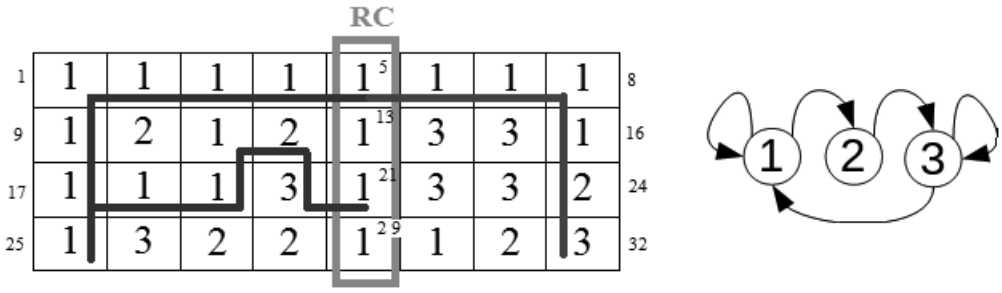


FIGURE 9.5 – Répartition des actions du robot : Exemple avec points potentiels d'ordre 1 et point critique

Au lancement de l'algorithme de décomposition, la vérification est lancée pour la recherche de points potentiels d'ordre 1. La fonction *CalculPP()* renvoie comme résultats : les points d'indice 5 et 21 de la colonne de recouvrement sont des points potentiels d'ordre 1. Les lignes en rouge sur la figure (figure 9.5) sont les chemins trouvés par *CalculPP()*.

La liste de points potentiels n'étant pas vide, la fonction *checkPP()* retourne `vrai`. La fonction *CalculPC()* est donc exécutée pour trouver les points critiques. Cette dernière trouve que 5 est un point critique d'ordre 1. Les lignes en bleu sur la figure (figure 9.5) sont les chemins trouvés par *CalculPC()*. La liste de points critiques n'est donc pas vide, la fonction *checkPC()* retourne `vrai` : notre problème est résolu et la requête Π est vérifiée sur G.

9.4.2 Exemple 2 : Résolution avec points potentiels d'ordre 1 sans point critique

Pour l'exemple 2, nous avons proposé la répartition des actions décrite dans la figure 9.6 et nous utilisons les mêmes conventions graphiques et de notation que dans l'exemple précédent.

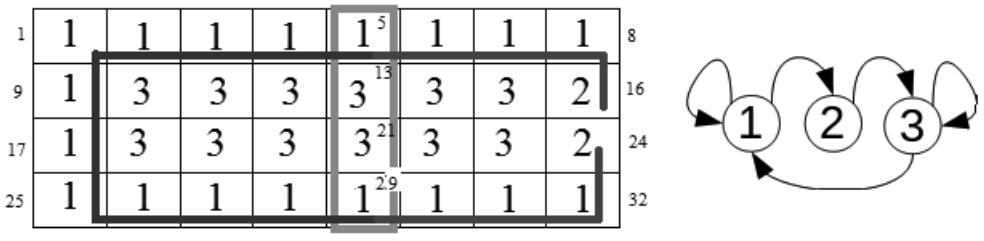


FIGURE 9.6 – Décomposition en vérification : Exemple avec points potentiels d'ordre 1 sans point critique

La vérification est lancée pour la recherche de points potentiels d'ordre 1. La fonction *CalculPP()* donne comme résultats : les points d'indice 5 et 29 sont des points potentiels d'ordre 1. Les lignes en rouge sur la figure (figure 9.6) sont les chemins trouvés par *CalculPP()*.

La fonction *checkPP()* retourne `vrai`, la fonction *CalculPC()* est donc exécutée. Mais il n'existe pas de point critique unique reliant toutes les cases du bord gauche de la grille à celles du bord droit, *checkPC()* retourne donc `faux`. Les lignes en bleu sur la figure (figure 9.6) sont les chemins trouvés par *CalculPC()*. On appelle alors la fonction *RepartitionG2()* qui va chercher à déterminer si toutes les cases du bord droit sont accessibles depuis un ensemble de points potentiels. Dans notre cas, le point d'indice 5 permet d'atteindre les points d'indice 8 et 16 du bord droit et le point d'indice 29 permet d'atteindre les points d'indice 24 et 32. La fonction *CheckRepartitionG2()* retourne donc `vrai` (tous les points de $\mathcal{B}_{\mathcal{D}}(G)$ sont atteignables) et par conséquence le problème d'atteignabilité Π est résolu sur G.

9.4.3 Exemple 3 : Résolution avec points potentiels d'ordre 2 et point critique

L' exemple 3 étudie la répartition des actions présentée dans la figure 9.7.

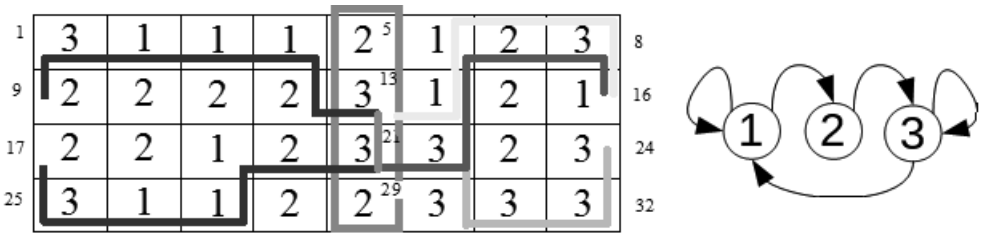


FIGURE 9.7 – Répartition des actions du robot pour l'exemple 3

La vérification est lancée pour la recherche de points potentiels d'ordre 1. On appelle la fonction *CalculPP()* qui retourne comme résultat : il n'existe aucun point potentiel d'ordre 1. On cherche donc maintenant des points potentiels d'ordre 2. Les lignes en rouge sur la figure (figure 9.7) sont les chemins trouvés par *CalculPP()*, et le chemin en vert entre 13 et 21 peut être parcouru dans les deux sens. La fonction *CalculPP()* retourne alors 13 et 21 comme points potentiels d'ordre 2 et la fonction *checkPP()* retourne `vrai`.

La fonction *CalculPC()* est donc maintenant exécutée pour identifier les points critiques s'il en existe. Cette dernière retourne que 13 n'est pas un point critique et que, par contre, 21 en est un. En effet, pour le point 13, les points 8 et 16 peuvent être atteints directement de 13 (chemin jaune dans la figure 9.7). En revanche, les chemins permettant d'atteindre 24 et 32 à partir de 13 passent forcément par un

autre point de la colonne de recouvrement (21). Or la case d'indice 21 a déjà été visitée dans la sous-grille S1, pour les chemins permettant d'atteindre 13 à partir de 17 et 25. Ainsi, les chemins permettant d'atteindre 24 et 32 ne sont donc plus autorisés en raison du principe d'unicité de visite. Pour le point 21, par contre, 8 et 16 peuvent être atteints via les points 22, 14, 6, 7, 8 et 16 (chemin violet dans la figure 9.7) et les points 24 et 32 peuvent être atteints via 22, 30, 31, 32 et 24 (chemin rose dans la figure 9.7). Par conséquence, 21 est donc bien un point critique.

Notre problème d'atteignabilité bord à bord de la grille est donc résolu avec un point critique à l'ordre 2.

9.4.4 Exemple 4 : Pas de point potentiel

La figure 9.8 présente un exemple de cas où il n'existe pas de point potentiel, et donc où notre algorithme ne trouve pas de solution. Les chemins en rouge sont les chemins explorés par la fonction *CalculPP()* à l'ordre 1.

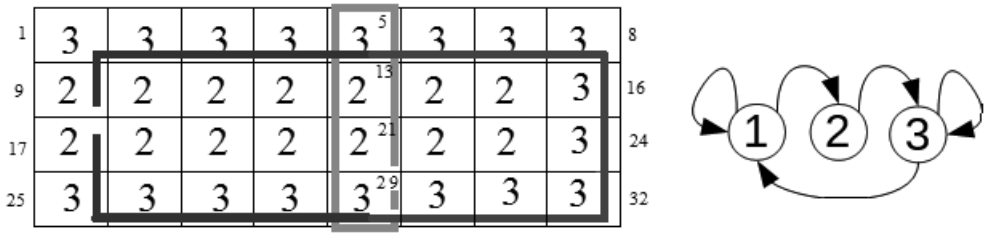


FIGURE 9.8 – Répartition des actions du robot pour l'exemple 4

Le point d'indice 5 est atteignable par les points d'indices 1 et 9 et le point d'indice 29 est atteignable par les points d'indices 17 et 25. Les chemins passant par le point d'indice 5 sont complètement indépendants des chemins passant par 29, et il n'est pas possible d'atteindre 5 à partir de 17 et 25 et de même pour 29 à partir de 1 et 9. 5 et 29 ne sont donc pas des points potentiels d'ordre 1. On répète l'appel de la fonction *CalculPP()* à l'ordre 2, 3, 4 et lorsque la variable *order* vaut 5, l'algorithme de décomposition se termine sans trouver de solution à la propriété d'atteignabilité Π.

Or si on avait appliquer la fonction *CalculPC()* pour les points d'indice 5 et 29, on aurait trouvé les chemins de couleur bleue : les points d'indices 5 et 9 permettent chacun d'atteindre tous les points du bord droit. Ainsi, il existe des chemins partant des points d'indices 1 et 9 et permettant d'atteindre tous les points du bord droit en passant par le point de la colonne de recouvrement d'indice 5 ; de même il existe des chemins partant des points d'indices 17 et 25 et permettant d'atteindre tous les points du bord droit en passant par 29.

Et donc, la propriété Π est vérifiée sur G, mais notre algorithme ne la détecte pas. Ce cas illustre la différence entre la version implémentée de notre algorithme, qui ne traite qu'une vérification par répartition partielle, et notre méthodologie théorique qui traite de la répartition complète.

9.5 Résultats et discussion

Dans cette section, nous étudions la performance de notre algorithme de décomposition pour différents scénarios et nous le comparons avec l'algorithme de référence (algorithme 2) qui cherche à résoudre le problème d'atteignabilité en employant UppAal directement sur la grille complète sans faire appel à une approche de décomposition.

9.5.1 Réduction de la complexité théorique

On peut distinguer 2 types de complexité au moment de la vérification de la propriété étudiée : la complexité en temps d'exécution et la complexité en espace mémoire consommée. Dans la partie expérimentale, les deux types de complexités sont étudiées.

Dans ce paragraphe, nous nous intéressons principalement à la réduction de la complexité en espace mémoire. En effet, l'explosion combinatoire se produit au moment de la vérification de la propriété étudiée parce qu'il ne reste plus d'espace mémoire disponible pour la vérification.

Pour évaluer la complexité en espace mémoire, nous considérons le nombre d'états générés dans le pire cas par le model-checking pour résoudre le problème d'atteignabilité étudiée. Comme déjà expliqué section 6.5, dans notre contexte, ce nombre d'états va essentiellement dépendre des dimensions de la grille et du nombre de mouvements possibles depuis une case. Pour la décomposition comme pour une approche plus classique, lorsque plusieurs propriétés sont vérifiées séquentiellement sur le même modèle, la complexité de l'espace mémoire globale se base sur la pire complexité de vérification d'une propriété. Dans la suite, une comparaison entre le nombre d'états engendré par la vérification sur la grille complète (référence) et celui obtenu en découpant la grille est fournie. Cette comparaison permet d'avoir une idée sur le facteur de réduction de la complexité en espace mémoire.

Théoriquement, l'algorithme de décomposition permet de réduire la complexité en espace mémoire d'un facteur $f = \frac{a^{wl}}{2 \times a^{wl/2}}$ avec a le nombre maximal de mouvements possibles à partir d'une case de la grille (dans notre cas $a = 3$), w la largeur de la grille et l la longueur de la grille. En effet, pour une grille de taille $w \times l$, le nombre d'états possibles N_G est de l'ordre de a^{wl} (cf section 6.5). Par exemple, pour une grille 4×16, $N_G = 3.4 \times 10^{30}$ et $f = 9 \times 10^{15}$ et pour une grille 4×32, $N_G = 1.17 \times 10^{61}$ et $f = 1.7 \times 10^{30}$. Pour la décomposition, on peut simplifier

la complexité de l'algorithme en considérant qu'il est de l'ordre de $2 \times a^{wl/2}$ car l'algorithme de décomposition fonctionne sur deux demi-grilles.

En ce qui concerne la complexité en temps d'exécution, l'algorithme de décomposition vérifie des propriétés d'atteignabilité sur des grilles plus petites mais le nombre de propriétés à vérifier est plus grand. Pour l'algorithme de décomposition, la complexité en temps d'exécution dépend de la complexité de la solution du problème de vérification (ordre 1, ordre 2, par point critique ou par répartition, etc). La complexité réelle étant difficile à établir, et à priori très différente de la complexité théorique pire cas que nous pourrions évaluer, nous nous limiterons dans la suite à une comparaison empirique des complexités.

9.5.2 Étude pratique de la complexité en espace et en temps

A) Principes pratiques de l'analyse et base de test

Comme pour le chapitre 6, nous avons choisi d'appliquer notre méthode de décomposition en vérification à des grilles de dimensions 4×16 (taille I) et 4×32 (taille II). Les actions affectées à chacune des cases permettent de définir des grilles représentatives de différentes classes de scénario en fonction de la solution de vérification (No solution, Ordre 1, Ordre 2), à savoir des grilles où la propriété II :

- n'est pas vérifiée (5 grilles de taille I et 5 de taille II),
- est vérifiée à l'ordre 1 (6 grilles de taille I, et 5 de taille II),
- est vérifiée à l'ordre 2 (5 grilles de taille I uniquement).

Ces grilles ont été construites grâce à une méthode ad-hoc qui mélange des cases avec des actions prédéterminées et des cases avec des actions choisies au hasard. Cette solution a été implémentée car la génération d'actions uniquement aléatoire conduisait la majeure partie du temps à des grilles sans solution pour la propriété d'atteignabilité bord à bord.

A ces différentes configurations s'ajoutent les deux cas particuliers du "All A" pour lequel l'action A est affectée à toutes les cases de la grille et qui représente donc la pire complexité puisque tous les chemins sont envisageables et le cas de "All B" pour lequel l'action B est affectée à toutes les cases de la grille et qui représente donc le cas le moins complexe car aucun chemin n'est possible.

Les résultats expérimentaux présentés dans la table 2.2 ont été obtenus avec le model checker d'UppAal en mode Breadth First Search (BFS). Ce dernier a été utilisé car il permet une comparaison équitable avec l'algorithme de référence. Comme nous l'avons déjà expliqué au chapitre 6, la propriété II est vérifiée sur une feuille du graphe d'états et la solution est toujours éloignée de la racine du graphe d'états car elle se trouve uniquement au bord droit de la grille. Le mode de recherche BFS

devrait présenter moins de variation de performances entre les requêtes pour une grille donnée (cf section 2.5).

Afin d'analyser les résultats expérimentaux obtenus, il est utile de considérer les points suivants :

- L'algorithme de référence vérifie toujours 4^2 propriétés d'atteignabilité sur des grilles de taille 4×16 et 4×32 (cf section 6.5).
- L'algorithme de décomposition vérifie des propriétés d'atteignabilité sur des grilles plus petites mais le nombre de propriétés à vérifier est plus grand. Par exemple, à l'ordre 1, pour une grille 4x16, entre 20 et 32 propriétés doivent être adressées. En effet en décomposant le problème on doit traiter 16 propriétés d'atteignabilité dans la sous-grille S1 (une pour chaque couple de points bord de gauche / colonne de recouvrement) et entre 4 à 16 propriétés d'atteignabilité pour la sous-grille S2 suivant le nombre de points potentiels trouvés. La solution du problème à des ordres plus importants nécessite de prendre en considération encore plus de propriétés.
- L'outil UppAal utilise une technique de recherche à la volée (*on the fly*), il s'arrête donc dès qu'une solution est trouvée.
- La taille de l'espace d'états dépend de la distribution des actions dans la grille (chapitre 6). Le cas "allA" est le scénario le plus gourmand en mémoire car tous les chemins sont possibles.
- Au moment de la vérification, UppAal alloue un ensemble minimum de blocs de mémoire même si la vérification effective pourrait consommer moins de mémoire pour les différents cas d'études analysés.

Le processus d'analyse et certaines spécificités de l'outil et/ou de la démarche ayant été rappelés, nous allons maintenant présenter et analyser les résultats expérimentaux obtenus en nous intéressant tout d'abord à la quantité de mémoire utilisée.

B) Complexité expérimentale en espace mémoire utilisé

La table 9.1 présente les résultats expérimentaux obtenus sur nos benchmarks avec un PC doté d'un processeur intel E5 cadencé à 3.2 GHz et doté de 256 Goctets de RAM.

Dans cette table, les 3 premières colonnes présentent les caractéristiques des benchmarks utilisées. A savoir les dimensions de la grille analysée, la classe de scénario et la référence du benchmark étudié (taille.n°) [3]. Les 4ème et 5ème colonnes

3. Remarque : les références ici ne correspondent pas à celles du tableau 6.1, la répartition des actions dans les grilles étant différentes.

indiquent la quantité de mémoire, exprimée en kBytes (Koctets) utilisée pour résoudre le problème d'atteignabilité en faisant appel à l'algorithme de référence, ou à la méthode de décomposition. Enfin la dernière colonne calcule, en pourcentage, le gain de la mémoire obtenu en faisant appel à notre approche de décomposition. Un nombre négatif traduit le fait que la méthode de décomposition proposée s'est révélée plus coûteuse que l'approche globale.

TABLE 9.1 – Comparaison en Mémoire (M) entre l'algorithme de Référence et l'algorithme de Décomposition

Grille	Classe	Scénarios	Référence	Décomposition	Gain
			M(KB)	M(KB)	%
Taille I	Ordre 1	I.1	13 856	6 516	53%
		I.2	7 464	6 252	16%
		I.3	12 184	6 452	47%
		I.4	12 548	6 512	48%
		I.5	21 456	9 340	56%
		I.all A	1 835 732	17 080	99%
	Ordre 2	I.1	8 320	6228	25%
		I.2	6 416	76	99%
		I.3	76	76	0%
		I.4	6 280	76	99%
		I.5	6 288	76	99%
	no solution	I.1, I.all B	76	76	0%
		I.2	8 388	76	99%
		I.3	76	5 968	$< 0\%$
		I.4	4 088	76	98%
Taille II	Ordre 1	II.1	5 473 520	15 808	100%
		II.2	2 280 964	13 172	99%
		II.3	17 548 756	39 116	100%
		II.4	36 800 592	28 352	100%
		II.all A	*>130 Go*	3 756 720	–
	no solution	II.all B	76	76	0%
		I.1	9 000	6 844	24%
		I.2	12 420	12 316	1%
		I.3	27 564	27 452	0%
		I.4	87 144	86 904	0%

Pour toute la base de test, il apparaît que l'algorithme de décomposition réussit globalement à réduire les besoins en mémoire de manière importante, mais de façon non uniforme. Nous analysons ci-dessous ces résultats.

Il existe un ensemble de scénarios où les gains sont faibles ($\leq$25%) voire inexis-

tants (0%). Ces faibles performances peuvent avoir deux origines. Soit la complexité de résoudre Π sur la grille complète était très faible et appliquer l'approche de décomposition était inutile. Cette explication s'applique aux scénarios ayant une complexité initiale si faible que leurs besoins en mémoire étaient inférieurs à l'allocation initiale minimum de mémoire de l'outil UppAal (76 KB), ce qui ne permet pas de comparaison de la mémoire réellement utilisée (0% de gain). Soit l'algorithme de décomposition a des exigences de mémoire comparables à celles de l'algorithme de Référence. C'est le cas des scénarios I.2 de l'ordre 1 de taille I et I.1 de l'ordre 2 de taille I. Ce cas est plus problématique. Il correspond à des situations pour lesquelles la complexité initiale était condensée dans une des demi-grilles. Dans ce cas, la décomposition en demi-grilles n'apporte rien. On peut donner comme exemple une grille avec seulement des actions A sur toutes les cases des colonnes de gauches, puis uniquement des actions B sur les colonnes de droite. Ceci dit, cela signifie également que si la complexité initiale de ces cas se limite à une demi grille, la complexité globale ne sera jamais proche du pire cas.

Le scénario I.3 de la classe "No Solution" de taille I est un cas spécial car l'algorithme de décomposition a augmenté la complexité pour ce scénario qui était initialement très faible. L'algorithme de décomposition vérifie plusieurs requêtes d'atteignabilité pour un scénario et la complexité globale se base sur la pire complexité. Or dans certains cas avec décomposition, la requête d'atteignabilité consiste à visiter un point de la colonne de recouvrement sans passer par un autre point de cette colonne. Cette requête est complexe et trouver un chemin la vérifiant nécessite que ce chemin vérifie plusieurs contraintes de visite et cela entraîne par conséquent l'exploration d'une plus grande partie de l'espace d'état, d'où l'augmentation de la complexité. Il faut noter que ceci dépend fortement de la répartition des actions dans la grille.

Pour la majorité des scénarios de l'ordre 1 de taille I, les gains sont acceptables et intéressants (>40%). Ce comportement est celui attendu, du fait que l'algorithme de décomposition vérifie des propriétés sur des grilles plus petites donc la vérification devrait être moins complexe. Pour ces cas de complexité "moyenne", la répartition des actions dans la grille influe évidemment sur le nombre de chemins qui existe dans chaque sous-grille mais reste dans un comportement "moyen", et donc avec un gain normal.

Pour le cas "all A", qui est le pire des cas pour l'ordre 1, et pour certains autres scénarios de taille 1, la réduction est spectaculaire (> 98%). De même, les 5 scénarios de taille II (grilles de taille 4x32) ayant une solution ont été testés afin de démontrer l'effet de la complexité exponentielle en fonction de la taille de la grille, et l'effet de la décomposition dans ce cas. Si on exclue le cas extrême allA, la mémoire requise pour l'algorithme de référence se situe entre 2 Go et 37 Go, tandis que l'algorithme de décomposition peut résoudre tous les cas avec une mémoire inférieure à 40 Mo. L'utilisation mémoire grossit exponentiellement avec la taille de la grille, ce qui donne une réduction également exponentielle : les gains sont toujours $\geq 99\%$.

Enfin, nous atteignons les limites pratiques et physiques de résolution du problème d'atteignabilité avec notre ordinateur pour le scénario II.all A. Comme nous l'avons déjà mentionné au chapitre 6, le cas II.allA atteint 130 GB au bout de 33 jours sans avoir terminé la vérification de la grille complète. Par contre, notre approche de vérification par décomposition permet elle aisément d'adresser des grilles de tailles supérieures.

Ainsi, nous avons montré que l'algorithme de décomposition réussit à réduire drastiquement les besoins en mémoire et permet de traiter des grilles de plus grandes tailles. Le gain en complexité dépend de la complexité initiale et donc de la taille de la grille, de la répartition des actions au sein de la grille, et de la complexité de résolution de la solution (ordre 1 ou supérieur).

Après avoir analysé la complexité expérimentale en espace nous allons maintenant nous intéresser à la complexité expérimentale en temps des algorithmes employés.

C) Complexité expérimentale en temps : Durée d'exécution

La table 9.2 présente les temps d'exécution enregistrés sur nos benchmarks. Les 3 premières colonnes présentent les caractéristiques des exemples utilisés. Les 4ème et 5ème colonnes indiquent la durée d'exécution exprimée en secondes observée pour résoudre le problème d'atteignabilité en faisant appel à l'algorithme de référence et à la méthode de décomposition. On distingue le Temps d'Exécution (TE) nécessaire à la vérification de toutes les propriétés du problème, du temps moyen TR nécessaire pour la vérification d'une seule propriété. Pour l'algorithme de Référence, le nombre de propriétés vérifiées est toujours égal à 16. Pour l'algorithme de décomposition, le nombre de requêtes de vérification est donné dans la colonne (N). Enfin, la dernière colonne calcule, en pourcentage, le gain de temps obtenu, si il existe, en faisant appel à notre approche de décomposition. Un nombre négatif traduit le fait que la méthode de décomposition proposée s'est révélée plus coûteuse que l'approche globale.

Les résultats que nous obtenons montrent des temps de calcul bien supérieurs pour la méthode de décomposition par rapport à l'approche de référence, pour les scénarios d'ordre 2 de taille I et pour tous les scénarios de la classe "no solution" (Taille I et Taille 2). Une analyse plus approfondie permet d'expliquer les raisons induisant ce comportement. En effet, comme nous l'avons souligné plus haut, par construction, la décomposition en vérification génère plus de requêtes de vérification que pour l'approche de référence, ce qui induit un temps de calcul supérieur à cette dernière. Les pires gains en temps d'exécution correspondent à un grand nombre de requêtes de l'algorithme de décomposition (entre 76 et 80), lié à la complexité de la solution (ordre 2). En revanche, la décomposition a toujours le temps de calcul moyen par requête (TR) le plus faible par rapport à l'approche de référence, ce qui prouve bien une baisse de la complexité locale.

TABLE 9.2 – Comparaison en Temps d'Exécution (TE) entre l'algorithme de Référence et l'algorithme de Décomposition

Grille	Classe	Scénarios	Référence		Décomposition			Gain
			TE (s)	TR (s)	TE (s)	N	TR (s)	%
Taille I	Ordre 1	I.1	5,51	0,34	2,88	20	0,14	48%
		I.2	2,91	0,18	2,25	20	0,11	23%
		I.3	4,46	0,28	2,74	20	0,14	39%
		I.4	4,77	0,30	2,15	20	0,11	55%
		I.5	5,86	0,37	3,03	20	0,15	48%
		I.all A	357,94	22,37	4,99	20	0,25	99%
	Ordre 2	I.1	2,11	0,13	7,82	76	0,10	-271%
		I.2	2,07	0,13	7,79	76	0,10	-276%
		I.3	1,74	0,11	7,71	76	0,10	-343%
		I.4	1,74	0,11	7,64	76	0,10	-339%
		I.5	1,97	0,12	7,69	76	0,10	-290%
	no solution	I.1, I.all B	1,6	0,10	1,61	16	0,10	-1%
		I.2	2,55	0,16	8,05	80	0,10	-216%
		I.3	1,65	0,10	1,72	16	0,11	-4%
		I.4	1,62	0,10	1,61	16	0,10	1%
Taille II	Ordre 1	I.1	2 234,67	139,67	5,51	20	0,28	100%
		I.2	1 075,70	67,23	5,01	20	0,25	100%
		I.3	7 116,84	444,80	7,23	20	0,36	100%
		I.4	15 308,00	956,75	7,99	20	0,40	100%
		I.all A	33 Jours	–	1 039,81	20	51,99	–
	no solution	II.all B	1,61	0,10	1,60	16	0,10	1%
		I.1	3,27	0,20	3,30	16	0,21	-1%
		I.2	7,87	0,49	7,43	16	0,46	6%
		I.3	17,91	1,12	18,00	16	1,13	-1%
		I.4	58,96	3,69	59,23	16	3,70	0%

Au contraire, on observe sur cette table que pour tous les scénarios d'ordre 1 (taille I et taille II), l'approche de décomposition offre un temps de calcul TE beaucoup plus petit que l'approche de référence (gain en temps d'exécution entre 23% et 100%). De plus, dès que la complexité du problème s'accentue (All A), et lorsque la grille croît en dimensions (taille II), les gains en temps d'exécution deviennent supérieurs à 99%.

En conclusion, notre algorithme de décomposition arrive à réduire le temps de résolution de la solution de vérification dans la plupart des cas. Dans certains cas, le temps d'exécution est plus mauvais, ce qui était attendu. Cependant, le temps

d'exécution reste tolérable ($< 10s$ pour les grilles de taille I). Nous pouvons donc considérer que notre solution de décomposition est utilisable dans tous les cas.

9.6 Conclusion

Le chapitre 6 qui s'intéressait à la résolution par model-checking du problème d'atteignabilité bord à bord d'une grille nous avait confronté au problème d'explosion combinatoire inhérent à ce type d'approche. Pour répondre à cette limitation, ce chapitre a proposé une méthodologie de décomposition basée sur le principe générique de décomposition. La décomposition en modélisation consiste à décomposer la grille globale G en deux sous-grilles ($S1$ et $S2$) de façon à avoir entre elles une colonne de recouvrement (RC) qui appartient à ces deux sous-grilles. La décomposition en vérification consiste à décomposer la propriété d'atteignabilité totale bord à bord (Π) en deux propriétés représentant des propriétés d'atteignabilité d'un bord à l'autre pour chacune des sous-grilles. Deux types de décomposition de la vérification sont proposés : la décomposition par point critique consiste à chercher dans $S1$ un point de RC permettant d'atteindre tous les points du bord gauche (Π_1), et dans $S2$ un point de RC permettant d'atteindre tous les points du bord droit (Π_2). L'analyse de la décomposition consiste à prouver que vérifier (Π_1) sur $S1$ et vérifier (Π_2) sur $S2$ permet de vérifier (Π) sur G sous certaines conditions. La décomposition par répartition applique les mêmes concepts mais en considérant la possibilité que plusieurs points de RC soit impliqués dans la solution globale.

L'algorithme de décomposition qui permet la mise en œuvre de l'approche de décomposition a été décrit et expliqué à travers des exemples d'exécution. Il faut noter que notre algorithme de décomposition de la vérification est basé sur la présence de points potentiels. S'il n'existe aucun point potentiel, notre algorithme de décomposition ne résout pas le problème même s'il existe une solution. La décomposition par répartition décrite dans l'approche de décomposition (cf section 9.2.2.2) permet de résoudre ce problème mais dans la version implémentée courante de notre algorithme (*AlgDécVerif()*), seule la répartition sur la grille $S2$ a été implémentée.

Les résultats expérimentaux que nous avons mené sur un ensemble de benchmarks montrent que notre méthode de décomposition est efficace par rapport à l'algorithme de Référence décrit dans le chapitre 6. Les gains en temps d'exécution sont hétérogènes, et dépendent de la complexité de résolution de la solution par notre algorithme de décomposition. Cependant, les performances en mémoire démontrent que la démarche de décomposition en vérification proposée permet d'effectuer des gains significatifs pour résoudre le problème d'atteignabilité bord à bord d'une grille de dimension conséquente.

De plus, le principe de décomposition pourrait être appliqué récursivement, ce qui permettrait de passer à l'échelle de grandes grilles. La méthodologie de décomposition pourrait également être développée en divisant, a priori, la grille initiale en

n sous-grilles (S1, .., Sn) au lieu de 2. Cette approche améliorée appliquerait une technique de décomposition similaire à celle présentée dans ce chapitre, pour S1 et Sn. Ensuite, elle chercherait un chemin entre un point potentiel u sur le bord droit de S1 et un point potentiellement critique v sur le bord gauche de Sn.

10 Conclusions et perspectives

10.1 Principaux résultats du mémoire

Nous nous sommes intéressés dans cette thèse à étudier des problématiques liées à l'agriculture de précision (AP). Le principe de l'AP est de rechercher et mettre en œuvre la Bonne action au Bon moment et au Bon endroit (les "3B"). Un état de l'art sur les fondamentaux de l'AP a été présenté dans le chapitre 1. L'agriculture de précision peut être vue comme une stratégie de gestion spatialisée et d'amélioration cyclique des procédés culturaux basée sur 5 fonctions qui sont l'observation, la caractérisation, la préconisation, l'application et le référencement spatial. La simulation constitue une alternative précieuse aux tests en situation réelle pour valider la commande d'une opération agricole. Cette approche est cependant limitée par la difficulté à prendre en compte en simulation des évènements non contrôlables. De plus, pour obtenir une simulation déterministe, la partie commande doit être spécifiée de façon complète : cette spécification peut conduire à sous-évaluer les possibilités de l'équipement et les marges d'optimisation de l'opération. Pour dépasser ces limites, l'emploi des méthodes formelles, et en particulier du model-checking, est a priori pertinent. Les bases du model-checking ont été présentées dans le chapitre 2. Le processus de model-checking se compose alors en 3 phases : la modélisation du système, la description de la spécification et enfin la vérification proprement dite. Enfin un chapitre consacré au projet adap2E précise le contexte de robotique agricole dans lequel se situent nos contributions.

Dans la partie 2, nous avons utilisé les techniques de model-checking pour modéliser, vérifier et, dans certains cas, optimiser des problèmes issus de l'agriculture de précision. Trois problèmes ont été traités : la pulvérisation de précision au sein d'un rang de vigne, la récolte viticole sélective suivant la qualité du raisin au champ, le parcours d'un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques. Tous ces problèmes ont en commun la gestion de la composante spatiale, qui se retrouve selon les trois représentations complémentaires : 1D pour la pulvérisation, 1D+ pour la récolte et 2D pour le parcours d'un espace ouvert.

Dans le chapitre 4, nous avons proposé la méthode AMPS qui permet de vérifier l'atteignabilité d'un objectif de traitement d'un rang de vigne, et de calculer une séquence de commande optimale pour la pulvérisation de ce rang à partir de données LiDAR. Du point de vue des fonctions de l'AP, la méthode AMPS couvre les 3 fonctions : caractérisation (dans l'étape 1 de la méthode), préconisation (étape 2) et application (étape 3). Elle met en œuvre le model-checking pour les fonctions préconisation et application. Après avoir décrit notre méthodologie, nous l'avons appliquée sur les données réelles issues d'un rang de vigne d'une parcelle du domaine Mas Piquet. Nous avons pu obtenir des premiers résultats intéressants. Cependant le calcul de la carte de commande optimale à haute résolution (inférieure au mètre) posait un problème d'explosion combinatoire. Pour pallier ce problème, nous avons proposé au chapitre 8 une approche de décomposition permettant de résoudre le problème sur tout un rang de vigne.

Dans le chapitre 5, nous avons proposé une approche permettant d'adresser le problème de vendange sélective (DHP - Differential Harvest Problem). On cherche alors à calculer une commande de récolte optimale (ici la commande décrit une logistique) à partir d'une carte de préconisation spatiale qui distingue deux qualités de raisins : la meilleure et l'ordinaire. L'objectif est de calculer une commande qui permet de récolter au moins une quantité $Rmin$ de raisins de la meilleure qualité, de respecter la carte de préconisation spatiale et de minimiser la durée totale de la vendange. Ce problème avait déjà été étudié dans la littérature en utilisant des techniques d'optimisation sous contraintes, nous l'avons abordé par model checking. Dans ce manuscrit nous avons modélisé les contraintes du DHP sous la forme d'un réseau d'automates temporisés de coût et nous avons également défini la propriété d'accessibilité qui permet de résoudre le problème posé. La vérification de cette propriété sur le modèle conçu avec l'outil UppAal-CORA permet de générer le contrôleur ayant le coût minimal (dans notre cas, la durée de récolte la plus courte) et vérifiant la propriété d'accessibilité tout en respectant toutes les contraintes du problème. Ce contrôleur optimal représente alors la logistique de récolte optimale. Comme dans le cas de la pulvérisation, pour un problème de taille réaliste, le problème d'explosion combinatoire est là encore rapidement rencontré (au delà de 9 rangs). L'utilisation de la fonctionnalité de "remaining" d'UppAal-CORA a permis de résoudre le DHP pour une parcelle de 12 rangs. Au delà, le problème d'explosion combinatoire persiste. Du point de vue des fonctions de l'AP, notre méthode de résolution par des techniques de model-checking du problème de DHP contribue à l'optimisation de la phase d'application de la carte de préconisation.

Dans le chapitre 6, nous avons étudié le parcours d'un robot autonome dans un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques. L'environnement agricole est représenté comme une grille spatiale rectangulaire régulière 2D. Dans chaque case de la grille, le robot effectue une seule action. Le robot est soumis à plusieurs contraintes : des contraintes de déplacement, des contraintes de visite et des contraintes de précédence entre les actions. Le fonctionnement du robot

a été modélisé sous la forme d'un réseau d'automates temporisés. Le problème de vérification étudié est celui de l'atteignabilité d'un bord de la grille à l'autre. On rencontre là encore un problème d'explosion combinatoire à partir d'une taille de grille de 32 colonnes et 4 lignes. Le chapitre 9 propose une approche de décomposition permettant de résoudre ce problème.

Comme souvent, l'utilisation du model-checking sur des exemples de complexité réelle conduit à des problèmes d'explosion combinatoire mis en évidence sur les 3 problèmes traités dans la partie II. Pour y faire face, une approche générique de décomposition a été proposée dans la partie III. Elle se décline de la façon suivante :

- Décomposition de la modélisation : partitionner le modèle initial du problème en n sous-modèles de sorte à avoir une zone de recouvrement entre 2 sous-modèles consécutifs ;
- Décomposition de la vérification : décomposer la résolution de la propriété à vérifier sur le modèle initial en la résolution de propriétés sur les sous-modèles ;
- Analyse de la décomposition : démontrer que la vérification des sous-propriétés sur chacun des sous-modèles permet de conclure sur la vérification de la propriété initiale sur le modèle global initial.

Ce principe a été employé sur deux des trois problèmes étudiés dans le mémoire. La mise en œuvre du principe de décomposition sur l'exemple de l'optimisation de la pulvérisation de précision permet de vérifier la propriété d'atteignabilité sur un rang de vigne complet de dimension quelconque et ainsi trouver la séquence de contrôle optimale. La décomposition en modélisation consiste à décomposer le rang complet en n parties de façon à avoir entre elles un bloc de recouvrement ayant un type spécifique sans végétation appelé "Trou". Ce type de décomposition est appelé "décomposition orientée trou". La décomposition en vérification consiste à vérifier une propriété d'atteignabilité partielle permettant d'obtenir la commande optimale sur chaque partie du rang ce qui permet de calculer la séquence et les coûts optimaux pour chaque partie. En cas d'absence ou d'insuffisance de trous de végétation, nous avons proposé une approche de décomposition complémentaire, appelée décomposition par injection, qui consiste à considérer comme blocs de recouvrement des blocs dont la commande locale est connue a priori. L'analyse de la décomposition a prouvé que l'union des séquences optimales des parties permet d'obtenir la séquence optimale complète et que la somme des coûts optimaux des parties permet d'obtenir le coût optimal global.

Le principe générique de décomposition a également été appliqué au problème de grille 2D afin de résoudre la question d'atteignabilité totale bord à bord. La décomposition en modélisation consiste à décomposer la grille globale G en deux sous-grilles ($S1$ et $S2$) ayant une colonne de recouvrement (RC). La décomposition en vérification consiste à décomposer la propriété initiale (Π) en deux propriétés (Π_1) et (Π_2). La propriété (Π_1) consiste à chercher des points de RC permettant

d'atteindre tous les points du bord gauche de la grille G et la propriété (Π_2) consiste à chercher un point de RC permettant d'atteindre tous les points du bord droit de la grille G. L'analyse de la décomposition consiste à prouver que vérifier (Π_1) sur $S1$ et vérifier (Π_2) sur $S2$ permet de vérifier (Π) sur G sous certaines conditions.

Nous avons prouvé empiriquement que notre approche apportais une nette amélioration des performances en utilisation mémoire en appliquant notre méthodologie sur un ensemble de scénarios d'utilisation sur des grilles de deux tailles différentes.

Nous venons de résumer les différentes contributions proposées dans ce manuscrit. Nous avons également montré que l'application des techniques de model-checking comme bien souvent lorsque l'on est confronté à des cas de dimensions réelles, doit faire face à un problème d'explosion combinatoire. Pour y faire face nous avons proposé et mis en œuvre une démarche générique de décomposition.

Nous présenterons dans la suite les limites et les perspectives de notre travail.

10.2 Limites et perspectives

Dans cette section, nous présenterons les limites et les perspectives spécifiques à chacune de nos contributions avant d'aborder plus globalement ces questions.

10.2.1 Optimisation de la pulvérisation

La méthode AMPS calcule une séquence de commande optimale pour la pulvérisation à partir de données LiDAR. Elle se base sur 3 étapes. Dans la suite, la limite de chaque étape sera décrite.

La première étape consiste à définir une fonction statique qui associe à chaque état de végétation deux commandes différentes et acceptables de pulvérisation C_{best} et C_{alt}. Ces commandes ont été définies en utilisant une expertise basée sur le test de pulvérisateurs réels sur un banc de vigne artificiel. Il serait pertinent de faire des tests sur le terrain pour valider ces commandes (fonction statique). Les tests à réaliser consisteraient dans un premier temps à appliquer une commande puis à calculer la quantité de produit interceptée par le végétal. Il faudrait dans un second temps s'assurer que cette quantité assure une protection suffisante. Par ailleurs il faudrait vérifier que la commande C_{best} est, pour une configuration de végétation donnée, une commande préférable à C_{alt}, soit parce que la quantité de produit pulvérisé est inférieure, soit parce que la qualité de répartition est meilleure. Ces tests ne sont pas du ressort des spécialistes du model-checking. L'UMR ITAP a une équipe en mesure de faire de tels tests.

L'étape 2 consiste à traiter les données LiDAR réelles pour obtenir une liste de blocs de végétation pour le rang étudié et pour chaque bloc on identifie sa durée de traitement et ses deux commandes C_{best} et C_{alt} en utilisant la fonction statique définie à l'étape 1. L'algorithme d'identification des blocs formant le rang pourrait

être amélioré. On rappelle que l'algorithme divise le rang en des blocs de densité homogène. Pour identifier la densité du bloc, la hauteur de végétation est découpée en 3 sections horizontales. Pour chaque section, 3 valeurs de densité sont considérées et ces valeurs sont obtenues en fixant des seuils sur le nombre des points d'interceptions. Ces seuils sont définis de façon empirique en se référant à des statistiques effectuées sur un seul rang de vigne. Il serait intéressant de développer une méthode générique permettant de définir ces seuils en se référant à des statistiques effectuées sur plusieurs rangs de vigne et dans différentes étapes du cycle végétatif de la vigne (débourrement, premières feuilles, floraison, fermeture des grappes, véraison). Les données LiDAR acquises récemment par l'UMR ITAP sur de nombreuses parcelles pourraient permettre de développer une telle méthode.

L'étape 3 consiste à produire une carte de commande du pulvérisateur, avec, pour chaque bloc de végétation identifié, sa durée de traitement et la configuration de pulvérisation retenue parmi C_{best} et C_{alt}. Cette carte résulte de la vérification d'une propriété d'atteignabilité sur le modèle PTA_AMPS. La dynamique du pulvérisateur (vitesse du pulvérisateur, temps d'ouverture et fermeture des buses, nombre de buses du pulvérisateur) est intégrée dans le modèle PTA_AMPS. Le modèle conçu est assez générique, et il peut être adapté pour tout type de pulvérisateur. Par exemple, si le nombre de buses change, il suffit d'ajuster le nombre d'instances de l'automate buse dans la partie déclaration du système et effectivement modifier la partie déclaration avec les nouvelles commandes C_{best} et C_{alt}. Même avec les limites de la version actuellement disponible de UppAal-CORA, il est a priori possible de réaliser l'étape 3 pour tout rang de vigne, en utilisant les méthodes de décomposition proposées dans ce mémoire.

Une perspective intéressante serait de comparer les performances de l'approche AMPS avec d'autres techniques de recherche de commande optimale comme par exemple la programmation dynamique.

Enfin, nos travaux n'ont de sens que s'ils peuvent être effectivement utilisés sur le terrain, par des agronomes, non rompus aux techniques de modélisation et au model-checking. Pour ce faire il faudrait intégrer la méthode AMPS dans un outil graphique et ergonomique qui, à partir des données LIDAR des rangs d'une parcelle de vigne et du choix des caractéristiques du pulvérisateur, déterminerait la séquence optimale de commandes de pulvérisation à appliquer.

La méthode AMPS que nous avons développée nous semble assez générique. Il serait sans doute intéressant de l'adapter à d'autres productions agricoles où les végétaux sont disposés en rangs de façon à évaluer les gains en pesticides pouvant être obtenus.

10.2.2 Vendange sélective

Le problème de récolte sélective consiste à chercher une logistique de récolte optimale à partir d'une carte de préconisation. Nous avons modélisé les contraintes du système sous forme d'un réseau d'automates temporisés de coût. Le modèle conçu permet la vérification d'une propriété d'atteignabilité permettant de déterminer la logistique optimale à mettre en œuvre. La limite principale de cette contribution est que notre approche ne permet de résoudre que des instances de 12 rangs. En effet, nous n'avons pas encore proposé de méthode de décomposition pour ce problème.

Plusieurs perspectives peuvent être envisagées.

La première consiste à renforcer l'utilité de notre approche en adaptant notre modèle à d'autres problèmes d'agriculture de précision impliquant le routage comme la récolte sélective pour d'autres cultures ou l'épandage de précision.

La deuxième consiste à modifier notre modèle pour couvrir plusieurs emplacements possibles pour les bennes. Cela aurait l'intérêt de minimiser la durée de trajet entre l'emplacement des bennes et les rangs non encore récoltés. Cela augmenterait la complexité du problème parce que le nombre de trajets possibles à parcourir par la machine serait multiplié par le nombre des emplacements possibles, ce qui justifierait d'autant plus l'intérêt du model checking pour trouver la meilleure logistique de commande.

Dans le cas général, la récolte se fait le matin. Les bennes doivent être entièrement remplies pour être transportées à la cave et l'objectif est de réduire la durée de ce stockage des raisins dans les bennes qui influe négativement sur la qualité des raisins (exposition à la chaleur, macération des grains dans le jus). Une autre vision de problème de vendange consiste donc à minimiser la durée de maintien du raisin récoltés dans les bennes avant leur transport à la cave. La stratégie de récolte consisterait à récolter les rangs de façon à remplir le plus rapidement possible les bennes.

De façon plus générale, ce problème de vendange sélective est très proche d'un problème de routage. Une approche similaire à la nôtre pourrait être utilisée pour optimiser les opérations robotiques dans les entrepôts.

10.2.3 Vérification d'une mission de robotique agricole

Notre dernière contribution est relative à la vérification d'une mission de robotique agricole. Le robot se déplace dans un environnement agricole modélisé par une grille 2D. Il effectue des opérations agricoles et il est soumis à plusieurs contraintes. Le fonctionnement du robot est représenté sous la forme d'un réseau d'automates. La mission du robot est exprimée sous la forme d'une propriété d'atteignabilité spécifique.

Une première limite de ce travail est que nous considérons un environnement agricole modélisé par une grille régulière ce qui est loin d'être le cas dans la réalité. Si la grille n'est pas régulière, la modélisation du problème sera plus complexe au plan technique. Cependant le travail réalisé dans le cas d'une grille rectangulaire serait une bonne base de départ et devrait pouvoir être généralisé aux grilles irrégulières.

Par ailleurs, notre proposition s'appuie actuellement sur un réseau d'automates non temporisés. Il serait bien évidemment pertinent d'ajouter explicitement une dimension temporelle dans notre modélisation, ce qui complexifierait sans doute la recherche de la propriété d'atteignabilité. Mais, l'ajout de cette dimension temporelle permettrait d'introduire un critère d'optimisation pour le choix des chemins, en ne retenant par exemple que ceux de durée "raisonnable", réduisant par la même la complexité du problème de recherche de propriété.

Toutes ces limitations peuvent a priori être levées sans grande difficulté.

Dans la section suivante, les limites de l'approche de décomposition seront abordées.

10.2.4 Limites de l'approche de décomposition

Une réponse à l'explosion combinatoire observée pour les travaux que nous avons développés a été de proposer une approche générique de décomposition.

Notre approche de décomposition a été appliquée avec succès à deux contributions : la pulvérisation de précision et la vérification d'une mission de robotique agricole.

En ce qui concerne la pulvérisation de précision, notre approche de décomposition n'est pas applicable sur des rangs où il n'y a pas beaucoup de blocs de type "trous" ou de blocs où $C_{best} = C_{alt}$. Une approche alternative de décomposition pourrait être étudiée.

Si l'on s'intéresse maintenant au problème de vérification d'atteignabilité bord à bord d'une mission de robotique agricole la méthode de décomposition de la grille initiale en deux sous-grilles n'est pas suffisante pour être applicable sur des grilles de grandes tailles. Dans ce cas, il faut précéder à une multi-coupe (découpage en n sous-grilles) ou appliquer l'approche de décomposition en deux sous-grilles de manière récursive. D'ailleurs le choix de la position de la colonne de recouvrement pourrait être étudié car il impacte certainement les performances de la décomposition en influençant le nombre de chemins générés dans chacune des sous-grilles. D'un point de vue général, le succès de l'approche de décomposition dépend de l'existence de points intermédiaires accessibles depuis un bord et permettant d'accéder au bord opposé de la grille, et de la facilité avec laquelle on peut les identifier (taille du chemin notamment). Leur existence permettrait de diviser le graphe en 2 ou plusieurs parties et de vérifier par décomposition l'atteignabilité. Nous sommes confiants dans

le fait qu'une théorisation graphe de l'approche de décomposition permettrait de généraliser l'approche que nous avons proposé.

Une perspective intéressante de notre approche de décomposition consiste à l'appliquer à d'autres problèmes de l'agriculture de précision ou de la robotique, et plus généralement à des problèmes impliquant des systèmes mobiles dans un espace. De plus, l'approche de décomposition a été étudiée dans ce mémoire uniquement pour une propriété d'atteignabilité, qui de plus est très spécifique. Or la force du model checking est justement de permettre de vérifier des propriétés plus complexes, comme par exemple étudier la décomposition pour des propriétés de sûreté.

Après avoir présenté les limites et perspectives spécifiques de chacune de nos contributions nous allons maintenant les analyser plus globalement.

10.2.5 Limites et perspectives générales

Si l'on considère l'ensemble de nos travaux un certain nombre de limitations et de perspectives communes peuvent être dégagées.

En premier lieu il nous semble pertinent de nous interroger sur la robustesse de nos modèles, c'est-à-dire sur leur capacité à s'adapter à des évolutions de configurations d'entrée ou à des perturbations. Certains des paramètres d'entrées peuvent être totalement génériques comme par exemple la taille d'un rang de vigne. D'autres peuvent être assez facilement étendus. Si l'on s'intéresse encore à nos travaux sur la pulvérisation, il semble relativement aisé d'étendre notre approche à des pulvérisateurs ayant plus ou moins de buses en définissant de nouvelles fonctions statiques. Enfin on peut constater que l'ensemble de nos travaux ne prennent en compte qu'une seule machine mobile. Il est évident qu'avec l'évolution de la robotisation il serait pertinent de développer et d'adapter nos modèles et nos méthodes à des problématiques multi-robots. Les modèles de fonctionnement des machines étudiées ne considèrent aucune perturbation telle qu'une buse défectueuse ou plus ou moins bouchée bien que ce type de phénomène soit loin d'être exceptionnel sur le terrain. De ce fait, le modèle actuel est incapable de gérer l'occurrence de tels dysfonctionnements. A terme il sera donc indispensable de construire et vérifier par model-checking un superviseur tolérant aux fautes. Bien évidemment, cela complexifiera d'autant le modèle élaboré et son analyse.

Par ailleurs, toutes nos études ont fait appel à l'outil de vérification formelle UppAal ou à sa version CORA pour la recherche de solution optimale en coût. Bien évidemment notre travail n'est pas intimement lié à un environnement de vérification particulier. Pour nos travaux, il pourrait être donc particulièrement intéressant de faire appel à d'autres outils de vérification formelle pour comparer les performances obtenues en temps de résolution et en mémoire utilisée.

Bien évidemment, comme tous les travaux relevant du model-checking et de la vérification, l'ensemble des approches que nous avons proposées ont été confrontées

au problème d'explosion combinatoire. Pour pallier cette importante limitation qu'il faut absolument surmonter pour adresser des questionnements de dimensions réelles nous avons naturellement présenté une approche de décomposition que nous avons appliquée dans le cadre de deux de nos contributions. Des perspectives intéressantes à l'approche de décomposition proposée peuvent être envisagées. D'une part nous nous sommes focalisés sur la propriété d'atteignabilité mais il serait sans doute pertinent d'aborder d'autres propriétés utiles dans les exploitations agricoles comme par exemple la sûreté. D'autre part, bien évidemment, nous pourrions envisager d'étendre notre travail sur la décomposition de la propriété d'atteignabilité à d'autres problèmes relevant de l'agriculture de précision, de la robotique ou, plus largement des systèmes mobiles.

Enfin, si l'on veut que ce type de travaux s'appuyant sur des modèles complexes et sur des outils de vérification sophistiqués pénètre le monde agricole il faut impérativement les mettre à la portée des utilisateurs finaux visés. En effet, l'analyse d'une trace de preuve de propriété n'est guère aisée pour un agriculteur. Il faut donc entamer un travail d'intégration des approches proposées au sein d'outils d'aide à la décision adaptés. Les données brutes d'entrée de tels systèmes peuvent être des images aériennes de parcelles, des données Lidar ou GPS, les caractéristiques des machines agricoles. Une interface homme-machine ergonomique et flexible doit permettre à l'utilisateur de restituer ces données, d'exprimer simplement ses requêtes et contraintes, et de visualiser clairement, après un temps d'attente acceptable, le résultat de l'analyse demandée. Ce travail de développement informatique est en cours de réalisation pour notre travail sur la pulvérisation de précision.

10.3 Conclusion générale

L'objectif principal de cette thèse était d'explorer la capacité des méthodes formelles, et en particulier des techniques de model-checking, à résoudre des problèmes issus de l'agriculture de précision. La question générale qui se pose donc à l'issue de ce travail est la suivante : "Est ce que le model-checking permet de répondre aux besoins de l'agriculture de précision ?". L'agriculture de précision se base sur un ensemble de fonctions, pour mettre en œuvre une stratégie qui commence par l'observation et finit par l'application, sur la base d'un référencement spatial. Cette stratégie a pour objectif d'appliquer la bonne action au bon moment et au bon endroit (les 3B). Ainsi, à un instant donné et à un endroit précis, il existe plusieurs actions possibles et il faut explorer de manière exhaustive tous les cas possibles pour trouver la bonne action. Nous avons besoin d'une part d'un formalisme qui permet de modéliser les actions, les contraintes temporelles et spatiales et d'une autre part d'une méthode qui permet de trouver la bonne action à appliquer. Une des voies est le model-checking. En effet, les actions, les positions spatiales, les contraintes temporelles, la dynamique du système dans le cas général peuvent être représentées en utilisant des automates temporisés. Il suffit alors de décrire le principe de 3B sous

la forme d'une propriété d'atteignabilité en logique temporelle. La vérification de cette propriété explore un espace d'états et fournit une trace d'exécution qui est une commande admissible. Cette commande permet de répondre aux besoins de l'agriculture de précision. Par exemple, si l'objectif de l'AP est de réduire les produits phytosanitaires, la synthèse du contrôleur, que le model-checking offre, permet de générer la trace de commandes de pulvérisation qui minimise la quantité du produit pulvérisée respectant les contraintes d'une protection suffisante à chaque endroit.

Ce manuscrit, au travers de plusieurs problématiques relevant de l'agriculture de précision, a démontré l'intérêt de recourir aux modèles et outils de model-checking pour répondre à certains questionnements des agriculteurs. Les contraintes dynamiques, spatiales et temporelles différent d'un problème à un autre mais l'objectif de l'agriculture de précision peut s'exprimer d'une façon unifiée : il s'agit de maximiser un profit (monétaire ou autre) ou de minimiser un coût, en respectant l'environnement et toutes les contraintes du système. Une courte annexe (annexe B) souligne les développements possibles dans le cadre d'analyse du model-checking CORA.

Notre technique a montré son efficacité à travers sa mise en œuvre sur les trois problèmes étudiés dans ce travail de thèse qui sont : la pulvérisation de précision au sein d'un rang de vigne, la récolte viticole sélective suivant la qualité du raisin au champ, le parcours d'un espace ouvert pour réaliser des actions agricoles sous contraintes spécifiques. Nous espérons que nos travaux, assez précurseurs, engendreront d'autres études qui confirmeront la pertinence de notre démarche.

Bibliographie

[1] Altisen, K., Bouyer, P., Cachat, T., Cassez, F., and Gardey, G. Introduction au contrôle des systèmes temps-réel. In Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05) (2005), Hermès, pp. 367–380.

[2] Alur, R., Courcoubetis, C., and Dill, D. Model-checking in dense real-time. Information and computation 104, 1 (1993), 2–34.

[3] Alur, R., and Dill, D. Automata for modeling real-time systems. In International Colloquium on Automata, Languages, and Programming (1990), Springer, pp. 322–335.

[4] Alur, R., and Dill, D. L. A theory of timed automata. Theoretical Computer Science 126 (Apr. 1994), 183–235.

[5] Alur, R., La Torre, S., and Pappas, G. J. Optimal paths in weighted timed automata. In International Workshop on Hybrid Systems : Computation and Control (2001), Springer, pp. 49–62.

[6] Amnell, T., Behrmann, G., Bengtsson, J., D'argenio, P. R., David, A., Fehnker, A., Hune, T., Jeannet, B., Larsen, K. G., Möller, M. O., et al. Uppaal-now, next, and future. In Summer School on Modeling and Verification of Parallel Processes (2000), Springer, pp. 99–124.

[7] Andersen, L. O. Program analysis and specialization for the C programming language. PhD thesis, University of Cophenhagen, 1994.

[8] Asarin, E., Maler, O., Pnueli, A., and Sifakis, J. Controller synthesis for timed automata. IFAC Proceedings Volumes 31, 18 (1998), 447–452.

[9] Bardin, S. Introduction au Model Checking ENSTA. 24 octobre 2008.

[10] Bastianelli, M., De Rudnicki, V., Codis, S., Ribeyrolles, X., and Naud, O. Two vegetation indicators from 2d ground lidar scanner compared for predicting spraying deposits on grapevine. In EFITA 2017 (2017), pp. 12–p.

[11] Bechar, A., and Vigneault, C. Agricultural robots for field operations : Concepts and components. Biosystems Engineering 149 (2016), 94–111.

[12] Behrmann, G., David, A., and Larsen, K. G. A tutorial on uppaal. In Formal methods for the design of real-time systems. Springer, 2004, pp. 200–236.

[13] Behrmann, G., David, A., Larsen, K. G., Håkansson, J., Pettersson, P., Yi, W., and Hendriks, M. Uppaal 4.0.

[14] Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., and Vaandrager, F. Minimum-cost reachability for priced timed automata. In International Workshop on Hybrid Systems : Computation and Control (2001), vol. 1, Springer, pp. 147–161.

[15] Behrmann, G., Fehnker, A., Hune, T., Larsen, K. G., Petterson, P., and Romijn, J. Guiding and cost-optimality in uppaal. In AAAI-Spring Symposium on Model-based Validation of Intelligence (2001), pp. 66–74.

[16] Behrmann, G., Larsen, K. G., and Rasmussen, J. I. Priced timed automata : Algorithms and applications. In FMCO (2004), vol. 3657, Springer, pp. 162–182.

[17] Behrmann, G., Larsen, K. G., and Rasmussen, J. I. Optimal scheduling using priced timed automata. ACM SIGMETRICS Performance Evaluation Review 32, 4 (2005), 34–40.

[18] Bellon-Maurel, V., and Huyghe, C. L'innovation technologique dans l'agriculture. Géoéconomie, 3 (2016), 159–180.

[19] Bengtsson, J. Memtime, 2002.

[20] Bengtsson, J., and Yi, W. Timed automata : Semantics, algorithms and tools. In Advanced Course on Petri Nets (2003), Springer, pp. 87–124.

[21] Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. Systems and software verification : model-checking techniques and tools. Springer Science & Business Media, 2013.

[22] Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. Comparison of the expressiveness of timed automata and time petri nets. In International Conference on Formal Modeling and Analysis of Timed Systems (2005), Springer, pp. 211–225.

[23] Berk, P., Hocevar, M., Stajnko, D., and Belsak, A. Development of alternative plant protection product application techniques in orchards, based on measurement sensing systems : A review. Computers and Electronics in Agriculture 124 (2016), 273 – 288.

[24] Bimbraw, K. Autonomous cars : Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In 2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO) (2015), vol. 1, IEEE, pp. 191–198.

[25] Bisgaard, M., Gerhardt, D., Hermanns, H., Krčál, J., Nies, G., and Stenger, M. Battery-aware scheduling in low orbit : the gomx–3 case. In FM 2016 : Formal Methods : 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings 21 (2016), Springer, pp. 559–576.

[26] Boucheneb, H., and Barkaoui, K. Covering steps graphs of time petri nets. Electronic Notes in Theoretical Computer Science 239 (2009), 155–165.

[27] Boucheneb, H., Lime, D., Parquier, B., Roux, O. H., and Seidner, C. Optimal reachability in cost time petri nets. In Formal Modeling and Analysis of Timed Systems (Cham, 2017), A. Abate and G. Geeraerts, Eds., Springer International Publishing, pp. 58–73.

[28] Bourely, A. Fruit harvest robotization : 10 years of cemagref experience on apple, grape and orange. In AgEng'90 Agricultural Engineering International Conference (1990), pp. 178–179.

[29] Bouyer, P. Weighted timed automata : Model-checking and games. Electronic Notes in Theoretical Computer Science 158 (2006), 3–17.

[30] Bouyer, P. Model-checking timed temporal logics. Electronic Notes in Theoretical Computer Science 231 (2009), 323–341.

[31] Bouyer, P., and Laroussinie, F. Vérification par automates temporisés. Systemes temps-réel 1, 121–150.

[32] Bouyer, P., and Laroussinie, F. Model checking timed automata. Modeling and Verification of Real-Time Systems : Formalisms and Software Tools (2008), 111–140.

[33] Bramley, R. Understanding variability in winegrape production systems 2. within vineyard variation in quality over several vintages. Australian Journal of Grape and Wine Research 11, 1 (2005), 33–42.

[34] Bramley, R., and Hamilton, R. Understanding variability in winegrape production systems : 1. within vineyard variation in yield over several vintages. Australian Journal of Grape and Wine Research 10, 1 (2004), 32–45.

[35] Briot, N. Programmation par contraintes pour les tournées en agriculture de précision. PhD thesis, 2017. Thèse de doctorat dirigée par Bessière, Christian et Vismara, Philippe Informatique Montpellier 2017.

[36] Briot, N., Bessiere, C., Tisseyre, B., and Vismara, P. Integration of operational constraints to optimize differential harvest in viticulture. In Proc. 10th European Conference on Precision Agriculture (ECPA 2015) (2015), pp. 487–494.

[37] Briot, N., Bessiere, C., and Vismara, P. A constraint-based approach to the differential harvest problem. In Proc. 21st International Conference on Principles and Practice of Constraint Programming (CP 2015), vol. 9255 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2015, pp. 541–556.

[38] Briot, N., Bessiere, C., and Vismara, P. Programmation par contraintes pour la vendange sélective. In Actes des Onzièmes Journées Francophones de Programmation par Contraintes (JFPC 2015) (2015), pp. 51–56.

[39] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L.-J. Symbolic model checking : 10^{20} states and beyond. Information and computation 98, 2 (1992), 142–170.

[40] Cassez, F., and Roux, O. H. Structural translation from time petri nets to timed automata. Journal of Systems and Software 79, 10 (2006), 1456–1468.

[41] Cassman, K. G., and Plant, R. E. A model to predict crop response to applied fertilizer nutrients in heterogeneous fields. Fertilizer research 31, 2 (Feb 1992), 151–163.

[42] Choi, K. H., Han, S. K., Han, S. H., Park, K.-H., Kim, K.-S., and Kim, S. Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields. Computers and Electronics in Agriculture 113 (2015), 266–274.

[43] Cimatti, A. Industrial applications of model checking. In Summer School on Modeling and Verification of Parallel Processes (2000), Springer, pp. 153–168.

[44] Clabaut, M., Ge, N., Breton, N., Jenn, E., Delmas, R., and Fonteneau, Y. Industrial grade model checking : use cases, constraints, tools and applications.

[45] Clarke, E. M., and Emerson, E. A. Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on Logic of Programs (1981), Springer, pp. 52–71.

[46] Clarke, E. M., Grumberg, O., and Peled, D. Model checking. MIT press, 1999.

[47] Clarke, E. M., and Wing, J. M. Formal methods : State of the art and future directions. ACM Computing Surveys (CSUR) 28, 4 (1996), 626–643.

[48] Codis, S., Carra, M., Delpuech, X., Montegano, P., Nicot, H., Ruelle, B., Ribeyrolles, X., Savajols, B., Vergès, A., and Naud, O. Dataset of spray deposit distribution in vine canopy for two contrasted performance sprayers during a vegetative cycle associated with crop indicators (lwa and trv). Data in brief 18 (2018), 415–421.

[49] Comparetti, A. Precision agriculture : Past, present and future. In International scientific conference AGRICULTURAL ENGINEERING AND ENVIRONMENT-2011 (2011), Aleksandras Stulginskis University, pp. 216–230.

[50] da Silva, M., and Magalhães, P. G. Modeling and design of an injection dosing system for site-specific management using liquid fertilizer. Precision Agriculture (2018), 1–14.

[51] Dechter, R. Constraint processing. Morgan Kaufmann, 2003.

[52] Dedousis, A., Godwin, R., O'Dogherty, M., Tillett, N., and Grundy, A. Inter and intra-row mechanical weed control with rotating discs. Precision Agriculture 7 (2007), 493–498.

[53] Desbourdes, C. Les utilisations du gps en agriculture. In TIC, robotique et télédétection en agricultures : les évolutions récentes, Séances hebdomadaires publiques de l'Académie d'Agriculture, p. séance du 17 décembre 2014.

[54] Desriers, M. L'agriculture française depuis cinquante ans : des petites exploitations familiales aux droits à paiement unique. Agreste cahiers 2 (2007), 3–14.

[55] Duflot, M., Kwiatkowska, M., Norman, G., Parker, D., Peyronnet, S., Picaronny, C., and Sproston, J. Practical applications of probabilistic model checking to communication protocols.

[56] Durrant-Whyte, H. A critical review of the state-of-the-art in autonomous land vehicle systems and technology. Albuquerque (NM) andLivermore (CA), USA : SandiaNationalLaboratories 41 (2001).

[57] Dusseux, P., Zhao, Y., Cordier, M.-O., Corpetti, T., Delaby, L., Gascuel-Odoux, C., and Hubert-Moy, L. Paturmata, a model to manage grassland under climate change. Agronomy for sustainable development 35, 3 (2015), 1087–1093.

[58] Emerson, E. A., and Halpern, J. Y. "sometimes" and "not never" revisited : on branching versus linear time temporal logic. Journal of the ACM (JACM) 33, 1 (1986), 151–178.

[59] Fahrenberg, U., Larsen, K. G., and Legay, A. Model-based verification, optimization, synthesis and performance evaluation of real-time systems. In Unifying Theories of Programming and Formal Engineering Methods. Springer, 2013, pp. 67–108.

[60] Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R., and Ouenzar, M. Comparison of model checking tools for information systems. In International Conference on Formal Engineering Methods (2010), Springer, pp. 581–596.

[61] Gao, G., Xiao, K., and Li, J. Precision spraying model based on kinect sensor for orchard applications.

[62] Gil, E., Escola, A., Rosell, J., Planas, S., and Val, L. Variable rate application of plant protection products in vineyard using ultrasonic sensors. Crop Protection 26, 8 (2007), 1287–1297.

[63] Godary, K. Validation temporelle de réseaux embarqués critiques et fiables pour l'automobiles. PhD thesis, 2004.

[64] Godefroid, P., Van Leeuwen, J., Hartmanis, J., Goos, G., and Wolper, P. Partial-order methods for the verification of concurrent systems : an approach to the state-explosion problem, vol. 1032. Springer Heidelberg, 1996.

[65] GRENIER, G. Robotique agricole : Retour vers le futur ! In TIC, robotisation en agriculture (2017), p. séance du 25 janvier 2017.

[66] Grumberg, O., and Veith, H. 25 years of model checking : history, achievements, perspectives, vol. 5000. Springer, 2008.

[67] Halder, R., Proença, J., MacEdo, N., and Santos, A. Formal verification of ros-based robotic applications using timed-automata. pp. 44–50.

[68] Hall, A., Louis, J., and Lamb, D. Characterising and mapping vineyard canopy using high-spatial-resolution aerial multispectral images. Computers & Geosciences 29, 7 (2003), 813–822.

[69] Helias, A., Guerrin, F., and Steyer, J.-P. Représentation par automates temporisés de contraintes temporelles-cas de la fertilisation organique des cultures de l'île de la réunion. In actes de la 4ème conférence de modélisation et simulation-organisation et conduite d'activité dans l'industrie et les services (MOSIM'03) (2003), pp. 691–698.

[70] Hélias, A., Guerrin, F., and Steyer, J.-P. Using timed automata and model-checking to simulate material flow in agricultural production systems - application to animal waste management. Computers and Electronics in Agriculture 63, 2 (2008), 183–192.

[71] Hendriks, M., Behrmann, G., Larsen, K., Niebert, P., and Vaandrager, F. Adding symmetry reduction to uppaal. In International Conference on Formal Modeling and Analysis of Timed Systems (2003), Springer, pp. 46–59.

[72] Herrera, D., Tosetti, S., and Carelli, R. Dynamic modeling and identification of an agriculture autonomous vehicle. IEEE Latin America Transactions 14, 6 (2016), 2631–2637.

[73] Hossard, L., Guichard, L., Pelosi, C., and Makowski, D. Lack of evidence for a decrease in synthetic pesticide use on the main arable crops in france. Science of the Total Environment 575 (2017), 152–161.

[74] Hélias, A. Agrégation/abstraction de modèles pour l'analyse et l'organisation de réseaux de flux : application à la gestion des effluents d'élevage à La Réunion. PhD thesis, 2003.

[75] Jones, J. L. Robots at the tipping point : the road to irobot roomba. IEEE Robotics & Automation Magazine 13, 1 (2006), 76–78.

[76] Jürschik, P., Coquil, B., Menzl, M., Fischier, C., Kersebaum, K., and Wegehenkel, M. Farmstar, new remote sensing services for farmers [farmstar, neue fernerkundungsdienste für landwirte]. VDI Berichte, 1855 (2004), 415–423. cited By 0.

[77] Kayacan, E., Kayacan, E., Ramon, H., and Saeys, W. Modeling and identification of the yaw dynamics of an autonomous tractor. 9th Asian Control Conference, ASCC 2013 (2013).

[78] Kilby, P., and Shaw, P. Chapter 23 - vehicle routing. In Handbook of Constraint Programming, F. Rossi, P. van Beek, and T. Walsh, Eds., vol. 2 of Foundations of Artificial Intelligence. Elsevier, 2006, pp. 801 – 836.

[79] Kirk, I. W., Fritz, B. K., and Hoffmann, W. C. Aerial methods for increasing spray deposits on wheat heads. In 2004 ASAE Annual Meeting (2004), American Society of Agricultural and Biological Engineers, p. 1.

[80] Koo, H.-M., and Mishra, P. Functional test generation using property decompositions for validation of pipelined processors. In Proceedings of the conference on Design, automation and test in Europe : Proceedings (2006), European Design and Automation Association, pp. 1240–1245.

[81] Lamb, D. W., Weedon, M., and Bramley, R. Using remote sensing to predict grape phenolics and colour at harvest in a cabernet sauvignon vineyard : Timing observations against vine phenology and optimising image resolution. Australian Journal of Grape and Wine Research 10, 1 (2004), 46–54.

[82] Lamport, L. Sometime is sometimes not never : On the temporal logic of programs. In Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (1980), ACM, pp. 174–185.

[83] Largouët, C. Aide à l'interprétation d'une séquence d'images par la modélisation du système observé. application à la reconnaissance de l'occupation du sol. PhD thesis, Thèse de doctorat, Université de Rennes I.[cité p. 155, 198], 2000.

[84] Largouët, C., and Cordier, M.-O. Improving the landcover classification using domain knowledge. INTERNATIONAL ARCHIVES OF PHOTOGRAMMETRY AND REMOTE SENSING 33, B4/2 ; PART 4 (2000), 538–545.

[85] Largouët, C., and Cordier, M.-O. Timed automata model to improve the classification of a sequence of images. In ECAI (2000), pp. 156–160.

[86] Largouët, C., Cordier, M.-O., Bozec, Y.-M., Zhao, Y., and Fontenelle, G. Use of timed automata and model-checking to explore scenarios on ecosystem models. Environmental Modelling & Software 30 (2012), 123–138.

[87] Larsen, K., Behrmann, G., Brinksma, E., Fehnker, A., Hune, T., Pettersson, P., and Romijn, J. As cheap as possible : effcient cost-optimal reachability for priced timed automata. In International Conference on Computer Aided Verification (2001), Springer, pp. 493–505.

[88] Larsen, K. G., Pettersson, P., and Yi, W. Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer 1, 1-2 (1997), 134–152.

[89] Lee, W. S., Slaughter, D., and Giles, D. Robotic weed control system for tomatoes. Precision Agriculture 1, 1 (1999), 95–113.

[90] Lenain, R, c. "anr adap2e project reference anr-14-ce27-0004 adaptive autonomous production platform for environment", 2014.

[91] Lherbier, J. Valorisation de l'information géographique en agriculture de précision. PhD thesis, 2005.

[92] Lomuscio, A., and Raimondi, F. The complexity of model checking concurrent programs against ctlk specifications. In International Workshop on Declarative Agent Languages and Technologies (2006), Springer, pp. 29–42.

[93] Lorber, F., Larsen, K., and Nielsen, B. Model-based mutation testing of real-time systems via model checking. pp. 59–68.

[94] Mazzanti, Franco et Ferrari, A. Ten diverse formal models for a cbtc automatic train supervision system. arXiv preprint arXiv : 1803.10324 (2018).

[95] McBratney, A., Whelan, B., Ancev, T., and Bouma, J. Future directions of precision agriculture. Precision agriculture 6, 1 (2005), 7–23.

[96] McBratney, A. B., and Taylor, J. A. Pv or not pv ? In Proceedings of the 5th international symposium on cool climate viticulture and œnology. Melbourne, Australia.

[97] Najem, M. Model-Checking symbolique pour la vérification de systèmes et son application aux tables de décision et aux systèmes d'éditions collaboratives distribuées. PhD thesis, École Polytechnique de Montréal, 2009.

[98] Naud, O., Verges, A., Hebrard, O., Codis, S., Douzals, J., and Ruelle, B. Comparative assessment of agro-environmental performance of vineyard sprayers using a physical full scale model of a vineyard row. In AgEng 2014 (2014), pp. 7–p.

[99] Pai, N., Salyani, M., and Sweeb, R. Regulating airflow of orchard airblast sprayer based on tree foliage density. Transactions of the ASABE 52, 5 (2009), 1423–1428.

[100] Parasuraman, R., Sheridan, T. B., and Wickens, C. D. A model for types and levels of human interaction with automation. IEEE Transactions on systems, man, and cybernetics-Part A : Systems and Humans 30, 3 (2000), 286–297.

[101] Pingault, N., Pleyber, E., Champeaux, C., Guichard, L., and Omon, B. Produits phytosanitaires et protection intégrée des cultures : l'indicateur de fréquence de traitement. Notes et études socio-économiques 32 (2009), 61–94.

[102] Pnueli, A. The temporal logic of programs. In Foundations of Computer Science, 1977., 18th Annual Symposium on (1977), IEEE, pp. 46–57.

[103] Queille, J.-P., and Sifakis, J. Specification and verification of concurrent systems in cesar. In International Symposium on programming (1982), Springer, pp. 337–351.

[104] Rasmussen, J. I., Larsen, K. G., and Subramani, K. Resource-optimal scheduling using priced timed automata. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2004), Springer, pp. 220–235.

[105] revised Version, A. Ametist deliverable 2.2. 2.

[106] Richards, D., and Stedmon, A. To delegate or not to delegate : A review of control frameworks for autonomous cars. Applied ergonomics 53 (2016), 383–388.

[107] Robert, P. L'agriculture de précision : les verrous liés à la technologie et à la gestion agronomique. In Colloque Agriculture de Précision, Educagri, Dijon (2000), pp. 11–29.

[108] Rosell, J. R., Llorens, J., Sanz, R., Arno, J., Ribes-Dasi, M., Masip, J., Escolà, A., Camp, F., Solanelles, F., Gràcia, F., et al. Obtaining the three-dimensional structure of tree orchards from remote 2d terrestrial lidar scanning. Agricultural and Forest Meteorology 149, 9 (2009), 1505–1515.

[109] Roudier, P., Tisseyre, B., Poilvé, H., and Roger, J.-M. A technical opportunity index adapted to zone-specific management. Precision agriculture 12, 1 (2011), 130–145.

[110] Saddem, R., Naud, O., Dejean, K. G., and Crestani, D. Decomposing the model-checking of mobile robotics actions on a grid. IFAC-PapersOnLine 50, 1 (2017), 11156–11162.

[111] Schnoebelen, P. The complexity of temporal logic model checking. Advances in modal logic 4, 393-436 (2002), 35.

[112] Sheridan, T. B., and Verplank, W. L. Human and computer control of undersea teleoperators. Tech. rep., Massachusetts Inst of Tech Cambridge Man-Machine Systems Lab, 1978.

[113] Sinfort, C., Cotteux, E., Bonicelli, B., and Ruelle, B. Une méthodologie pour évaluer les pertes de pesticides vers l'environnement pendant les pulvérisations viticoles. In STIC & Environnement (2009), pp. 14–p.

[114] Slaughter, D., Giles, D., and Downey, D. Autonomous robotic weed control systems : A review. Computers and electronics in agriculture 61, 1 (2008), 63–78.

[115] Solanelles, F., Escolà, A., Planas, S., Rosell, J., Camp, F., and Gràcia, F. An electronic control system for pesticide application proportional to the canopy width of tree crops. Biosystems engineering 95, 4 (2006), 473–481.

[116] Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., and Feyzeau, P. Path planning : A 2013 survey. Proceedings of 2013 Int. Conf. on Industrial Engineering and Systems Management, IEEE - IESM 2013 (2013).

[117] Tarjan, R. Depth-first search and linear graph algorithms. SIAM journal on computing 1, 2 (1972), 146–160.

[118] Tisseyre, B. Peut-on appliquer le concept d'agriculture de précision à la viticulture. Mémoire d'habilitation à diriger des Recherches. Soutenue le 10 (2012).

[119] Tokekar, P., Vander Hook, J., Mulla, D., and Isler, V. Sensor planning for a symbiotic uav and ugv system for precision agriculture. IEEE Transactions on Robotics 32, 6 (2016), 1498–1511.

[120] Ulrich, I., Mondada, F., and Nicoud, J.-D. Autonomous vacuum cleaner. Robotics and autonomous systems 19, 3-4 (1997), 233–245.

[121] van Schaik, S. J. Answering reachability queries on large directed graphs. PhD thesis, Department of Computing Sciences, Faculty of Science, Utrecht University, 2010.

[122] Xiongkui, H., Aijun, Z., Yajia, L., and Jianli, S. Precision orchard sprayer based on automatically infrared target detecting and electrostatic spraying techniques. International Journal of Agricultural and biological engineering 4, 1 (2011), 35–40.

[123] Yagoubi, R. S., Naud, O., Dejean, K. G., and Crestani, D. New approach for differential harvest problem : The model checking way. IFAC-PapersOnLine 51, 7 (2018), 57 – 63. 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[124] Zhao, Y. Modélisation qualitative des agro-écosystèmes et aide à leur gestion par utilisation d'outils de model-checking. PhD thesis, 2014. Thèse de doctorat dirigée par Cordier, Marie-Odile et Gascuel, Chantal Informatique Rennes 1 2014.

[125] Zhou, K., Jensen, A. L., Bochtis, D. D., and Sørensen, C. G. Simulation model for the sequential in-field machinery operations in a potato production system. Computers and Electronics in Agriculture 116 (2015), 173–186.

[126] Zion, B., Mann, M., Levin, D., Shilo, A., Rubinstein, D., and Shmulevich, I. Harvest-order planning for a multiarm robotic harvester. Computers and Electronics in Agriculture 103 (2014), 75–81.

[127] Zwaenepoel, P., and Le Bars, J.-M. L'agriculture de précision. Ingénieries-EAT, 12 (1997), p–67.

A Description détaillée de l'algorithme de décomposition pour grille 2D

L'algorithme décrit dans le chapitre 9 concerne la décomposition de la modélisation et la décomposition de la vérification d'une grille 2D.
Nous avons utilisé un organigramme pour décrire la structure globale de notre algorithme de décomposition (figure A.1). Les procédures et les traitements sont représentés par des rectangles. Les tests sont représentés par des losanges, le début et la fin sont représentés par des ellipses.

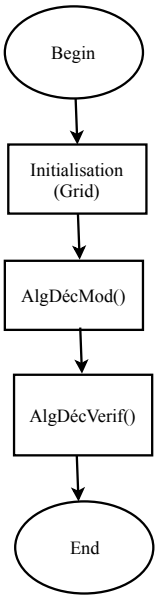


FIGURE A.1 – Organigramme de l'algorithme de décomposition

L'algorithme de décomposition est composé de 3 fonctions :

- La fonction *Initialisation(Grid)* : initialise les structures de données et les variables utilisées.

- La fonction *AlgDécMod()* : gère la décomposition de la modélisation.
- La fonction *AlgDécVerif()* : gère la décomposition de la vérification.

Dans la suite, la fonction *Initialisation()* est décrite.

A.1 La fonction *Initialisation()*

La fonction *Initialisation()* permet d'initialiser les structures de données et les variables nécessaire au déploiement de l'algorithme de vérification. Elle assure aussi la génération aléatoire des actions à réaliser par le robot dans la grille principale G (génération d'instances de test).

Dans la suite, on désigne par $X()$ la fonction qui permet d'initialiser la structure X. X peut être une liste ou une matrice.

Le pseudo code de la fonction *Initialisation()* est le suivant :

Algorithm 4 Initialisation()

Require: Les structures de données vides (RC, B_G, B_D, $GridRCB_G$, etc)
Ensure: Les structures de données remplies avec génération aléatoire des actions

1: **function** INITIALISATION($Grid$)
2: $InitVariable(largeur, longueur)$;
3: $RC()$; Initialisation de la liste RC
4: $B_G()$; Initialisation de la liste B_G
5: $B_D()$; Initialisation de la liste B_D
6: $GridRCB_D()$; Initialisation de la matrice $GridRCB_D$
7: $GridRCB_G()$; Initialisation de la matrice $GridRCB_G$
8: $GridB_DB_G()$; Initialisation de la matrice $GridB_DB_G$
9: $PP(largeur)$; Initialisation de la liste PP
10: PC; Initialisation de la liste PC
11: $GenerationActionOfRobot$
12: **end function**

$InitVariable(largeur, longueur)$: permet d'initialiser la longueur et la largeur de grille G. Par convention, la longueur de la grille G est le nombre de colonnes de la grille, et la largeur est son nombre de lignes.

Les structures initialisées sont comme suit :

RC est l'abréviation de Recovery column. RC est une liste et elle désigne l'ensemble des points de la colonne de recouvrement. Les points sont référencés par un indice établi selon la formule suivante : un point de coordonnée (x,y) a l'indice (x-1) * longueur + y avec longueur est la longueur de la grille G. Initialement, cette liste est vide et son remplissage sera effectué dans la fonction *AlgDécMod()*.

B_G (Bord de Gauche) est une liste. Elle contient les points du bord gauche de la grille. Les points sont référencés comme dans RC. *Exemple :* Dans la figure A.2, la longueur de la grille G est 8, la largeur de G est 4 et les indices du bord gauche sont $\{1, 9, 17, 25\}$.

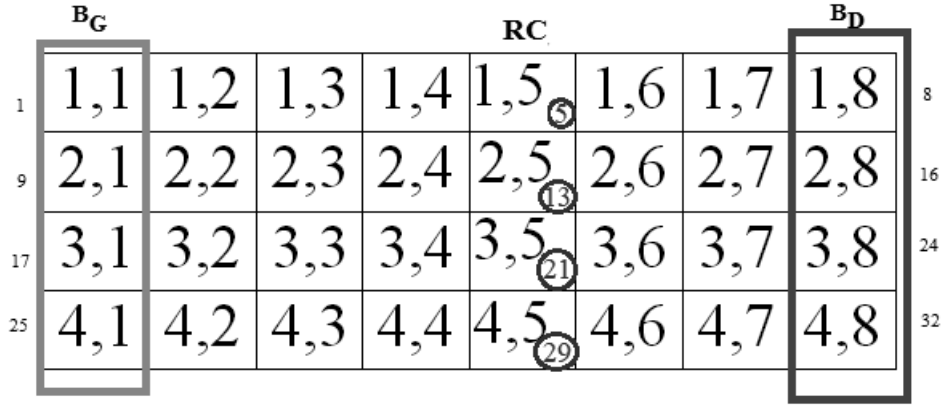


FIGURE A.2 – Représentation de la grille G avec visualisation des indices.

B_D (Bord de Droite) est une liste, elle contient les points du bord de droite de la grille. Chaque point de la liste est représenté par un indice, indice calculé de façon analogue aux fonctions $RC()$ et $B_G()$.
Exemple : Dans la figure A.2, les indices du bord de droite sont $\{8, 16, 24, 32\}$.

$GridRCB_G$ est une matrice dont les éléments sont des listes. Cette matrice sera utilisée dans la fonction *AlgDécVerif()*. Elle a pour dimensions $largeur$ x $largeur$, avec $largeur$ le nombre de lignes de la grille G (les nombres de points de RC et de B_G sont égaux à $largeur$). $GridRCB_G$ servira à mémoriser les points de passage dans la colonne de recouvrement en partant du bord de gauche. Cette matrice est initialisée à des listes vides.

$GridRCB_D$ est une matrice dont les éléments sont des listes et qui sera utilisée dans la fonction *AlgDécVerif()*. La taille de la matrice est égale à $largeur$x$largeur$. La matrice permet de mémoriser, pour un point donné de RC, les points de la colonne de recouvrement qui ne seront pas visités pour rejoindre un point du bord droit de la grille. $GridRCB_D$ est initialisée à des listes vides.

$GridB_DB_G$ est une matrice dont les éléments sont des entiers. La taille de cette matrice est égale à $largeur$x$largeur$. $GridB_DB_G(i, j)$ indique l'existence d'au moins un chemin entre i (bord de gauche) et j (bord de droite). Le calcul est mené en utilisant les matrices $GridRCB_G$ et $GridRCB_D$. $GridB_DB_G$ est une matrice dont tous les éléments sont initialisés à -1 et son remplissage sera effectuée dans la fonction *AlgDécVerif()*. Le code -1 signifie qu'il n'y a pas de chemin complet connu entre les deux points considérés (un point du bord de gauche et un point du bord de droite).

PP désigne Potential Point. C'est un tableau de liste de taille égale à $largeur$. PP[x] contient les points potentiels d'ordre x+1.

PC désigne Critical Point. C'est une liste de taille égale à $largeur$, initialement vide. Elle contient les indices des points critiques.

$GenerationActionOfRobot()$: Cette fonction permet de générer aléatoirement les actions à réaliser par le robot en chaque point de la grille. Elle les stocke dans un fichier `action.txt` sous forme des chiffres. On rappelle que dans notre cas, A, B et C sont les actions à effectuer par le robot dans le champ et ces actions sont soumises à des contraintes de précédence. Les actions A, B et C sont représentées dans le fichier `action.txt` respectivement par les chiffres 1, 2 et 3.

Une fois le processus d'initialisation terminé, la fonction *AlgDécMod()* décrite dans la section suivante est appelée.

A.2 Algorithme de décomposition de la modélisation

L'algorithme de décomposition de la modélisation *AlgDécMod()* commence par choisir une colonne de recouvrement se trouvant au centre de la grille ou à son voisinage. Ensuite, il génère le modèle des deux sous-grilles, à gauche et à droite de la colonne de recouvrement, sous la forme d'un fichier xml. Ces deux fichiers générés seront exécutés par l'outil UppAal en utilisant les fonctions memtime et verifyta.

Le pseudo code de la fonction *AlgDécMod()* est décrit par l'algorithme 5.

Algorithm 5 AlgDécMod()

Require: fichier `action.txt`, fichier xml de la grille G
Ensure: fichiers xml des sous grilles

```
1: function AlgDécMod(action.txt, fichier modèle.xml )
2:     FillRC();
3:     GenerationSubGridXML
4: end function
```

$FillRC()$: La fonction $FillRC()$ permet de remplir la liste RC. Chaque point appartenant à la colonne de recouvrement est représenté par un indice, calculé comme expliqué précédemment.
Exemple : Dans la figure A.2, les indices de la colonne de recouvrement sont $RC = \{5, 13, 21, 29\}$.

$GenerationSubGridXML(fichier\ action.txt,\ fichier\ modèle.xml)$: Cette fonction prend en entrée les fichiers contenant notre modèle et les actions de la grille globale G. Elle commence par décomposer, conformément à la décomposition en modélisation choisie, le contenu du fichier `action.txt` en deux fichiers `action1.txt`

et `action2.txt`. Le fichier `action1.txt` contient les actions à réaliser par le robot dans la sous-grille S_1 et le fichier `action2.txt` contient les actions à réaliser par le robot dans la sous-grille S_2. Ensuite, en utilisant le fichier `action1.txt` et le fichier xml du modèle (fichier modèle.xml), on génère le fichier xml relatif à la sous-grille S_1. De même pour la sous-grille S_2. Les deux fichiers xml générés seront exécutés par UppAal pour la recherche et la vérification des chemins.

Une fois l'algorithme de décomposition de la modélisation terminé, l'algorithme de la décomposition de la vérification décrit en section suivante est appelé.

A.3 Algorithme de la décomposition de la vérification

L'algorithme de la décomposition de la vérification proposé décompose la vérification en plusieurs phases. On commence par chercher dans la colonne de recouvrement les points potentiels (points atteignables à partir de tous les points de $B_G(G)$). Puis, on cherche à savoir si, parmi ceux-ci, certains sont des points critiques (points permettant d'atteindre tous les points du bord de droite de G). Enfin, s'il n'existe pas des points critiques, on vérifie si tous les points du bord de droite sont atteignables en utilisant un ensemble de points potentiels plutôt qu'un seul.

L'organigramme de la figure A.3 présente synthétiquement les différentes phases mises en œuvre pour la décomposition de la vérification d'une grille 2D. Il décrit le même algorithme que le pseudo-code fournit à l'algorithme 6.

L'algorithme de décomposition *AlgDécVerif()* est un algorithme itératif, composé principalement de 4 fonctions :

- La fonction *InitDecVerif()* : initialise les variables utilisées ($order$, $finish$ et $ResolvedProblem$).
- La fonction *CalculPP()* : calcule les points potentiels en travaillant sur la sous-grille S_1.
- La fonction *CalculPC()* : exploite la liste des points potentiels obtenue et calcule les points critiques en travaillant sur la sous-grille S_2.
- La fonction *RepartitionG2()* : exploite le résultat de la fonction *CalculPC()* et vérifie si tous les points du bord de droite sont atteignables en utilisant un ensemble de points potentiels plutôt qu'un seul.

Le pseudo-code de l'algorithme de décomposition *AlgDécVerif()* est décrit dans l'algorithme 6.

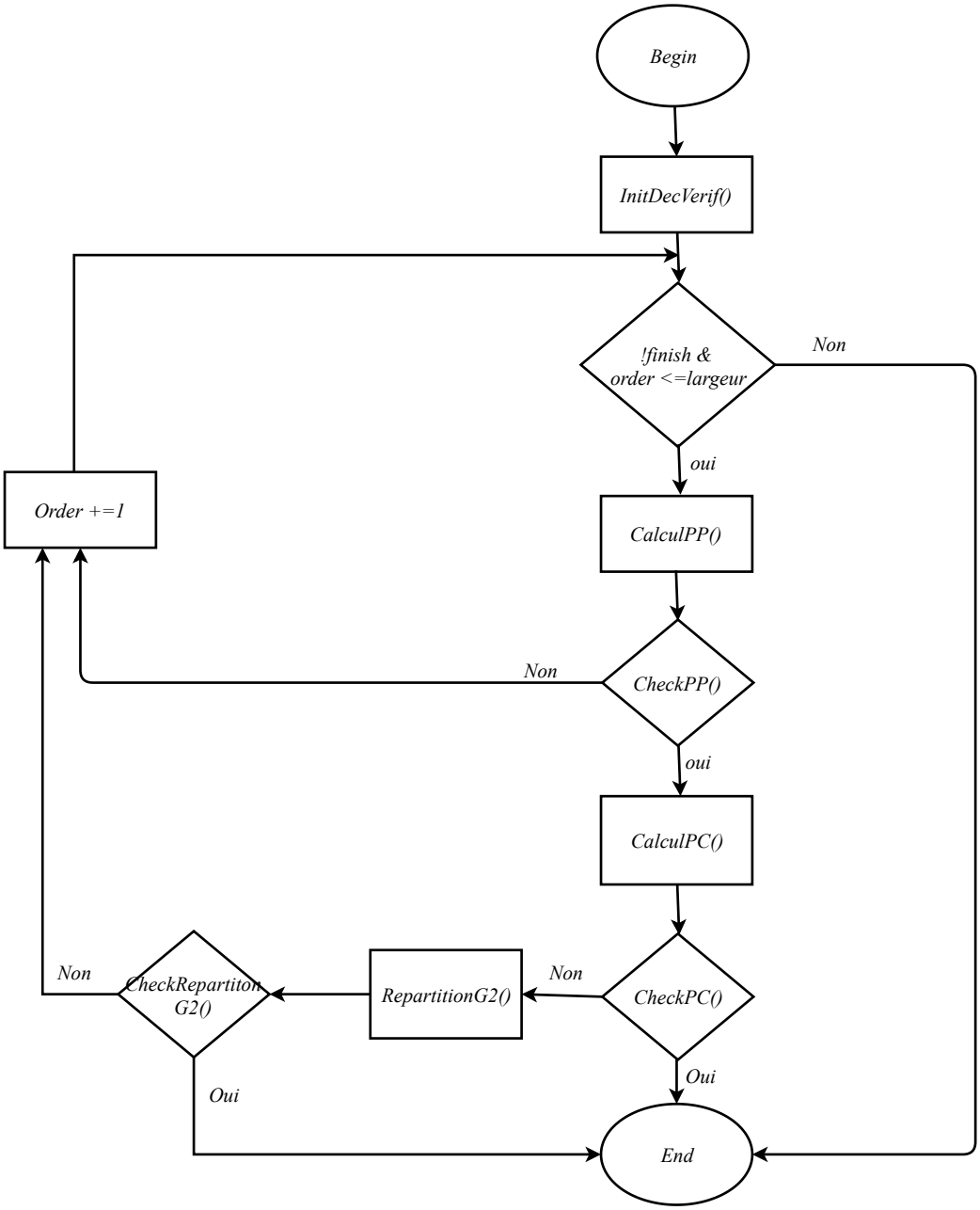


FIGURE A.3 – Organigramme de l'algorithme de décomposition

L'algorithme commence par appeler la fonction *InitDecVerif()*. Cette fonction initialise la variable *order* à 1, les variables $finish$ et $ResolvedProblem$ à faux. La variable *order* désigne l'ordre de point potentiel, la variable $finish$ vaut vrai lorsque le processus de vérification est terminé et la variable $ResolvedProblem$ désigne si la propriété Π est vérifiée ($ResolvedProblem$=vrai) ou non vérifiée ($ResolvedProblem$=faux).

Algorithm 6 AlgDécVerif()

```
1: procedure AlgDécVerif
2:     InitDecVerif() ;
3:     while finish = False and order ≤ largeur do
4:         CalculPP();
5:         if CheckPP() == False then
6:             order = order + 1;
7:         else
8:             CalculPC();
9:             if CheckPC() == True then
10:                finish = True;
11:                ResolvedProblem = True;
12:            else
13:                RepartitionG2();
14:                if CheckRepartitionG2() == True then
15:                    ResolvedProblem = True;
16:                    finish = True;
17:                else
18:                    order = order + 1;
19:                end if
20:            end if
21:        end if
22:    end while
23: end procedure
```

Ensuite, tant que la valeur de la variable $finish$ est faux et l'ordre *order* est inférieur à la largeur de la grille, l'algorithme *AlgDécVerif()* continue et il appelle la fonction *CalculPP()*. Cette dernière recherche les points potentiels d'ordre *order* et elle remplit la structure de données PP. On rappelle que PP est un tableau de liste de taille égale à $largeur$ et $PP[x]$ contient les points potentiels d'ordre x+1.

Si la liste $PP[order-1]$ est vide alors il n'existe pas des points potentiels d'ordre *order*. La fonction *CheckPP()* retourne vrai s'il existe des points potentiels d'ordre *order* ($PP[order-1]$!= vide) et retourne faux s'il en a pas ($PP[order-1]$ == vide). Si la fonction *CheckPP()* retourne faux alors on répète le processus en cherchant des points potentiels d'ordre supérieur. Ainsi, la valeur de la variable *order* s'incrémente. Si la fonction *CheckPP()* retourne vrai alors la fonction *CheckPC()* est appelée. Cette dernière exploite la liste $PP[order-1]$ pour rechercher les points critiques parmi les points potentiels trouvés et elle remplit la structure de données PC. On rappelle que PC est une liste de taille égale à $largeur$ qui contient les points critiques.

Si la liste PC est vide alors il n'existe pas des points critiques. La fonction *CheckPC()* retourne vrai s'il existe des points critiques (PC != vide) et retourne faux s'il en a pas (PC == vide).

Si la fonction *CheckPC()* retourne vrai alors l'algorithme se termine (les variables $finish$ et $resolvedproblem$ valent vrai). Ainsi, la propriété Π est vérifiée. Si la fonction *CheckPC()* retourne faux alors la fonction *CheckRepartitionG2()* est appelée. Cette dernière cherche si tous les points du bord de droite sont atteignables par un ensemble de points potentiels plutôt qu'un seul point potentiel. Si c'est le cas, la fonction *CheckRepartitionG2()* retourne vrai et l'algorithme se termine (les variables $finish$ et $resolvedproblem$ valent vrai). Ainsi, la propriété Π est vérifiée. Sinon, on répète le processus de recherche des points potentiels d'ordre supérieur. Ainsi, la valeur de la variable $order$ s'incrémente.

L'algorithme de décomposition *AlgDécVerif()* s'arrête lorsque $finish$ vaut vrai ou lorsque la valeur de la variable $order$ dépasse la largeur de la grille G.

On rappelle que la fonction *InitDecVerif()* initialise les variables $order$, $ResolvedProblem$ et $finish$ comme expliqué plus haut. Le reste des fonctions de l'algorithme de décomposition *AlgDécVerif()* seront détaillées dans les sections suivantes.

A.3.1 La fonction *CalculPP()*

La fonction *CalculPP()* calcule les points potentiels. La recherche de points potentiels se fait dans la sous-grille S_1. Elle consiste à trouver les points de la colonne de recouvrement tels que chaque point en question peut être atteint depuis n'importe quel point du bord gauche. On remarque que la fonction *CalculPP()* vérifie la propriété Π_1. Ainsi, la recherche des points potentiels est effectué à l'aide de l'outil UppAal, par une requête d'atteignabilité, sur le sous-modèle de S_1. Le pseudo code de la fonction *CalculPP()* est décrit dans l'algorithme 7.

Gestion des points potentiels d'ordre 1 (Algorithme 7, lignes 2 à 19)

On rappelle que l'ordre du point potentiel est le nombre de points de la colonne de recouvrement qui ont été visités pour atteindre ce point potentiel, point potentiel compris. Notre algorithme de décomposition donne la priorité à l'ordre 1 dans la recherche de points potentiels. Pour l'ordre 1, on commence par parcourir la liste RC.

La variable $Test$ permet de vérifier si un point potentiel $e1$ est atteignable par tous les points du $B_G(S_1)$. La fonction $len()$ permet de renvoyer la longueur d'une liste (nombre d'éléments).

La fonction *CallUppaal(ordre, source, destination, liste)*, appelée dans la fonction *CalculPP()* et dans la fonction *CalculPC()*, permet de :

- Calculer la requête à vérifier en partant de *source* vers *destination*. Le paramètre *liste* dépend de la fonction appelante. Si la fonction appelante est *CalculPP()*, alors le paramètre *liste* contient la destination et les points de la colonne de recouvrement qui doivent être visités pour atteindre la destination. Lorsque l'on est à l'ordre 1, seule la destination doit être visitée dans la colonne de recouvrement. Si la fonction appelante est *CalculPC()*, alors le paramètre *liste* contient les points de la colonne de recouvrement qui ne doivent pas être visités pour atteindre la destination.

- Écrire la requête calculée dans le fichier `query.q`.

- Créer un fichier `test.sh`. Dans ce fichier, on écrit la commande UppAal de vérification `verifyta`. Cette dernière prend comme paramètres la requête à vérifier (qui se trouve dans le fichier `query.q`) et le fichier xml de la sous-grille correspondante.

- Lancer `test.sh` et écrire le résultat dans le fichier `example.txt`.

Par exemple, si on est à l'ordre 1 et que l'on cherche un chemin entre le point d'indice 1 et le point d'indice 5, alors la fonction appelante est *CalculPP()*. Ainsi, la fonction *CallUppaal* sera *CallUppaal(1,1,5,[5])* et la requête générée est :
E<> i_init = 1 and j_init = 1 and i_fin = 1 and j_fin = 5 and visit[0][4]== true and visit[1][4]== false and visit[2][4]== false and visit[3][4]== false.
Autre exemple : si on est à l'ordre 2 et que l'on cherche un chemin entre le point d'indice 1 et le point d'indice 5, alors la fonction appelante est *CalculPP()*. Ainsi, la fonction *CallUppaal* sera *CallUppaal(2,1,5,[5,13])* et la requête générée est :
E<> i_init = 1 and j_init = 1 and i_fin = 1 and j_fin = 5 and visit[0][4]== true and visit[1][4]== true and visit[2][4]== false and visit[3][4]== false.
Dernier exemple : si on est à l'ordre 3 et que l'on cherche un chemin entre le point d'indice 5 et le point d'indice 8, alors la fonction appelante est *CalculPC()*. Ainsi, la fonction *CallUppaal* sera *CallUppaal(3,5,8,[13,21])* et la requête générée est :
E<> i_init = 1 and j_init = 5 and i_fin = 1 and j_fin = 8 and visit[1][4]== false and visit[2][4]== false .

$MAJGrid(GridRCB_G)$: Cette fonction permet d'analyser le résultat de la requête. Trois cas de figure sont possibles : i) la requête ne peut aboutir en raison de l'explosion combinatoire, ii) la requête n'est pas vérifiée, il n'existe pas de chemin entre la source et la destination respectant toutes les contraintes du problème, iii) la requête est vérifiée. En fonction de la réponse à la requête, la fonction met à jour la matrice de chemins $GridRCB_G$.
Exemple : Soient e_1 un point de RC et e_2 un point de B_G. Si la fonction *CallUppaal* retourne "requête vérifiée" et que l'on est à l'ordre 1, alors la fonction $MAJGrid$ ajoute e_1 à la grille $GridRCB_G$: $GridRCB_G[e_1, e_2] = e_1$.

Si e_1 est atteignable par tous les points du bord de gauche, alors la variable $Test$ reste à `true` et e_1 est un point potentiel : la fonction *CalculPP()* l'ajoute à la liste $PP[order - 1]$.

Une fois tous les chemins testés à l'ordre 1, on supprime tous les points potentiels d'ordre 1 trouvés de la liste RC. En effet, si e est un point potentiel d'ordre 1 alors cela ne sert à rien de trouver un chemin terminant à un ordre supérieur sur ce même point. Si tous les points sont des points potentiels d'ordre 1, alors RC devient l'ensemble vide, et l'algorithme de décomposition n'itérera pas à un ordre supérieur.

Si RC est vide après l'exécution de la fonction *CalculPP()*, alors, d'après l'organigramme de la fonction *AlgDécVerif()*, la fonction *CalculPC()* est appelée. Deux cas sont possibles : 1) il existe un point critique et le programme se termine, 2) il n'existe pas un point critique et la fonction de *RepartitionG2()* est appelée. Dans le deuxième cas, il y a deux possibilités : i) tous les points du bord de droite sont atteignables par un ensemble de points potentiels et l'algorithme se termine, ii) il y a des points du bord de droite qui ne sont pas atteignables par les points potentiels et dans ce cas la variable *order* s'incrémente. Si la variable $finish$ vaut faux l'algorithme continue l'exécution. Si e est un point potentiel d'ordre 1 alors cela ne sert à rien de faire de nouveaux tests pour ce point pour les ordres supérieurs à 1. En effet, si e est un point potentiel d'ordre 1 mais n'est pas un point critique, alors e ne sera pas un point critique à un ordre supérieur à 1. Par conséquent, si tous les points sont des points potentiels d'ordre 1 alors la variable $finish$ doit être mise à jour à la valeur *vrai* pour éviter des calculs inutiles.

Gestion des points potentiels d'ordre > 1 (Algorithme 7, lignes 20 à 41)

Si l'ordre des points potentiels est supérieur 1 alors plusieurs points de la colonne de recouvrement ont été visités avant d'atteindre la destination. Par exemple, l'ordre 2 signifie que 2 points de la colonne de recouvrement sont visités, destination comprise.

La condition $len(GridRCB_G(S_1)[e_1, e_2]) == 0$ or $order > 2$ permet de ne pas recalculer inutilement des points potentiels déjà traités à l'ordre 1.

La fonction $OptimisationCombination(order)$ permet de calculer l'ensemble des combinaison de points de RC pouvant être parcourus par un chemin donné, en fonction de l'ordre et les stocke dans la liste (de listes) *combinations*.

Exemple : Reprenons notre exemple d'illustration (voir figure A.2). Soit 5 l'indice du point destination dans la colonne de recouvrement, et supposons l'ordre égal à 2. Les combinaisons de visite possibles sont (5,13), (5,21) et (5,29). Sachant que chaque point de la grille n'est visité qu'une et une seule fois, les combinaisons (5,21) et (5,29) sont impossibles. Par exemple, pour la combinaison (5,21), il n'est pas possible de passer par 21 avant d'atteindre 5 sans passer par 13. Or l'ordre étant 2, il ne peut pas y avoir 3 cases visitées dans la colonne de recouvrement.

Le reste de la fonction *CalculPP()* fonctionne comme pour l'ordre 1.

Algorithm 7 Potential Points

```
1: function CalculPP   ▷ Fonction qui permet de calculer les points potentiels
2:     if order == 1 then
3:         for e_1 in RC do
4:             Test = True;
5:             for e_2 in B_G do
6:                 CallUppaal(order, e_2, e_1, list());
7:                 MAJGrid(GridRCB_G, e_2, e_1);
8:                 if len(GridRCB_G[e_1, e_2]) == 0 then
9:                     Test = False;
10:                end if
11:            end for
12:            if Test == True then
13:                PP[order − 1].add(e_1);
14:            end if
15:        end for
16:        RC = RC.remove(PP[order − 1]);
17:        if len(RC) == 0 then
18:            finish = True;          ▷ Car tous les points de RC ont été testés
19:        end if
20:    else
21:        for e_1 in RC do
22:            Test = True;
23:            for e_2 in B_G do
24:                if len(GridRCB_G[e_1, e_2]) == 0 or order > 2 then
25:                    combinations = OptimisationCombination(order)
26:                    for j in combinations do
27:                        if e_1 in j then
28:                            CallUppaal(order, e_2, e_1, j);
29:                            MAJGrid(GridRCB_G, e_2, e_1);
30:                        end if
31:                    end for
32:                end if
33:                if len(GridRCB_G[e_1, e_2] == 0 then
34:                    Test = False;
35:                end if
36:            end for
37:            if Test == True then
38:                PP[order − 1].add(e_1);
39:            end if
40:        end for
41:    end if
42: end function
```

Maintenant que nous avons déterminé tous les points potentiels de la colonne de recouvrement et leur ordre, en considérant la sous-grille de gauche ($S1$), nous allons maintenant nous intéresser aux points critiques pour la sous-grille de droite ($S2$).

A.3.2 La fonction *CalculPC()*

La fonction *CalculPC()* permet de calculer les points critiques. Un point critique est un point potentiel qui permet d'atteindre tous les points du bord de droite dans la sous-grille S_2. S'il existe un point critique dans S_2 notre problème est résolu et la propriété d'atteignabilité Π est vérifiée sur G. Le pseudo code de *CalculPC()* qui permet d'identifier les points critiques est décrit dans l'algorithme 8. Son principe de fonctionnement est le même que celui de la fonction *CalculPP()*. Dans *CalculPC()*, au lieu de parcourir la liste complète des points de RC, on parcourt uniquement la liste des points potentiels. De même, au lieu de modifier la grille $GridRCB_D$, on modifie $GridRCB_D$.

Il faut cependant prendre en considération qu'il peut ne pas exister de points critiques pour la sous grille de droite S_2, alors que, pour autant, il est possible d'atteindre tous les points du bord de droite à partir d'un point quelconque du bord de gauche. Pour traiter ce cas, nous allons faire appel à la fonction *RepartitionG2()*.

A.3.3 La fonction *RepartitionG2()*

En l'absence de la présence de points critiques dans la colonne de recouvrement, la fonction *RepartitionG2()* va chercher si tous les points du bord de droite peuvent être visités depuis un ensemble de points potentiels. En effet, il suffit qu'un sous-ensemble de tous les points potentiels permette d'atteindre tous les points du bord droit et Π sera vérifiée sur la grille G. Le pseudo code de la fonction *RepartitionG2()* est décrit dans l'algorithme 9.

Exemple : Supposons qu'après l'exécution de *CalculPC()* sur notre exemple, on trouve $RC = \{21, 29\}$, $PP[1] = \{5, 13\}$ et $PC = \emptyset$, ce qui signifie que 5 et 13 sont des points potentiels d'ordre 1 mais qu'ils ne sont pas des points critiques. De plus, on suppose que 5 permet d'atteindre 8 et 16 sur le bord de droite $B_D = \{8, 16, 24, 32\}$, et que 13 permet d'atteindre 24 et 32. Alors la totalité des points du bord droit peuvent être atteints à partir des points potentiels. Le problème est donc résolu : Π est vérifiée sur la grille G.

La fonction $MAJGridB_DB_G()$ remplit la matrice $GridB_DB_G$ en utilisant les matrices $GridRCB_G$ et $GridRCB_D$. Elle permet de vérifier l'existence de chemins du bord gauche au bord droit et par conséquent la vérification de la propriété Π sur la grille G. On rappelle que la grille $GridRCB_G$ stocke les points de RC traversés pour tous les chemins possibles entre un point de B_G et un point de RC. Au contraire, pour un même point de RC, la grille $GridRCB_D$ stocke les points de RC **non** traversés pour atteindre un point de B_D.

Algorithm 8 Critical Points

```
1: function CalculPC
2:     if order == 1 then
3:         for e_1 in PP[order − 1] do
4:             Test = True;
5:             for e_2 in B_D do
6:                 CallUppaal(order, e_1, e_2, list());
7:                 MAJGrid(GridRCB_D, e_1, e_2);
8:                 if len(GridRCB_D[e_1, e_2]) == 0 then
9:                     Test = False;
10:                end if
11:            end for
12:            if Test == True then
13:                PC.add(e_1);
14:            end if
15:        end for
16:    else
17:        for e_1 in PP[order − 1] do
18:            Test = True;
19:            for e_2 in B_D do
20:                combinations = OptimisationCombination(order)
21:                for j in combinations do
22:                    if e_1 in j then
23:                        CallUppaal(order, e_1, e_2, j);
24:                        MAJGrid(GridRCB_D, e_1, e_2);
25:                    end if
26:                end for
27:                if len(GridRCB_D[e_1, e_2]) == 0 then
28:                    Test = False;
29:                end if
30:            end for
31:            if Test == True then
32:                PC.add(e_1);
33:            end if
34:        end for
35:    end if
36: end function
```

L'idée est donc que pour un couple de points (e, f) donné, avec $e \in B_G$ et $f \in B_D$, s'il existe un chemin passant par RC qui est compatible entre les points de RC parcourus depuis la gauche et les points de RC non parcourus pour aller à droite, alors il existe un chemin complet de gauche à droite. L'existence de ce chemin est noté dans la

grille complète par : $GridB_DB_G[e, f] = 1$. Au contraire, si l'intersection est vide alors $GridB_DB_G[e, f] = -1$. Si, pour chaque couple (e, f), $GridB_DB_G[e, f] = 1$, alors il existe des chemins de tous les points de B_G à tous les points de B_D : la propriété Π est vrai sur G (dans l'algorithme, $ReparG2$ vaut vrai).

Exemple : Supposons que $GridRCB_G[1, 13] = \{\{13, 5\}, \{13, 21\}\}$, c'est à dire que pour aller de 1 à 13, il existe deux trajets. Le premier passe par les points d'indice 5 et 13 de la colonne de recouvrement et le deuxième passe par les points d'indice 13 et 21. Supposons maintenant que $GridRCB_D[13, 24] = \{13, 5\}$, c'est à dire que pour aller de 13 à 24, il existe un trajet qui **ne** passe **pas** par 5. On remarque que la liste $\{13, 21\}$ n'existe pas dans $GridRCB_D$[13,24], ce qui signifie qu'aller de 13 à 24 sans passer par 21 n'est pas possible. Au contraire, l'intersection des listes de$GridRCB_D$[13,24] et $GridRCB_G$[1,13] donne la liste $\{13, 5\}$. Il existe un chemin de 1 à 24 et ce chemin passe par 5, 13 et 21. On note donc $GridB_DB_G[1, 24] = 1$.

Algorithm 9 RepartitionGrid2

```
1: function RepartitionG2
2:     ReparG2 = True;
3:     MAJGridB_DB_G();
4:     while i < largeur do
5:         while j < largeur do
6:             if GridB_DB_G[i, j] == −1 then
7:                 ReparG2 = False;
8:             end if
9:             j = j + 1;
10:        end while
11:        i = i + 1;
12:    end while
13: end function
```

Nous avons identifié plusieurs voies d'optimisation pour améliorer notre algorithme de décomposition. Ces voies d'optimisation seront étudiées dans le chapitre perspective.

B Synthèse sur Model-checking et CORA

Le Model-checking permet de vérifier automatiquement si un modèle d'un système est conforme à ses spécifications. Pour ce qui est de l'accessibilité d'un état ou d'une propriété, il permet de vérifier cette accessibilité en répondant par oui ou par non. Retourner une trace de coût optimal n'est pas une fonctionnalité de base en model-checking. Par contre, une analyse CORA (Cost Optimal Reachability Analysis) représente le problème de trouver le coût minimum pour atteindre un état cible dans l'espace d'états de ce modèle, si cet état est accessible. Le principe de CORA est très intéressant et permet de combiner l'optimisation et la vérification. Il est dommage au plan technique pour la communauté scientifique que la version disponible de l'outil UppAal-CORA soit limitée par une compilation en architecture 32 bits.

Dans la littérature, le principe de CORA a été abordé pour les automates temporisés de coût [5, 14, 87]. Boucheneb et al [27] ont proposé une extension CORA pour les réseaux de Petri temporisés, et ils l'ont implémenté dans l'outil de model-checking Romeo. Ils ont illustré, à travers une étude de cas, l'utilité de cette approche pour modéliser les problèmes de consommation de ressources ou d'allocation de ressources dans des systèmes concurrents. Patricia Bouyer a aussi proposé une extension CORA pour les automates de jeu temporisé, appelée *Weighted timed games* [29]. La notion de coût est généralement utilisée dans un automate temporisé comme mesure de la performance des exécutions et elle peut être exprimée par une variable. Dans le contexte de la planification, il est possible de calculer le coût optimal pour atteindre un emplacement prédéterminé de l'automate. L'exécution de l'automate déterminée par l'algorithme CORA permettant d'atteindre un état avec coût optimal est un contrôleur optimal. Dans le contexte de la théorie des jeux [8, 1], il existe deux joueurs : le contrôleur et l'environnement. Les actions réalisées par le contrôleur sont des actions contrôlables et les actions réalisées par l'environnement sont des actions non contrôlables. Les règles de jeu peuvent être modélisés par un réseau d'automates de jeu temporisé. Une stratégie gagnante est une stratégie permettant de trouver une exécution menant le réseau d'automates à un état vérifiant une propriété souhaitée. Dans le cas où il existe plusieurs stratégies gagnantes, une approche CORA

permettant de chercher des stratégies gagnantes optimales. Une extension de notre travail vers les automates de jeu temporisé de coût pourrait être intéressante. Nous sommes convaincus que l'approche CORA pourrait être généralisée pour les modèles formels exprimant une notion de coût. Elle permet une synthèse de contrôleur optimal très utile pour l'optimisation décisionnelle sur les procédés, et en particulier pour les questions d'agriculture de précision.