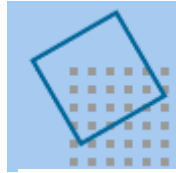


UM2



LIRMM



Mémoire de stage de Master :
Spécialité Recherche en Informatique
Mention Informatique, Mathématiques, Statistiques

Model-checking et ordonnancement :
Application à la décision de protection
phytosanitaire de la vigne

Version avant soutenance
Soutenance prévue le 20 juin 2007

Florent Hernandez

Tuteur de stage : Olivier Naud
Tuteur LIRMM : Rodolphe Giroudeau

Département Écotechnologies et Agrosystèmes

Unité Mixte de Recherche ITAP
Information et Technologies pour les Agro-procédés

Domaine de Lavalette
361, Jean François Breton ; B.P. 5095
34196 MONTPELLIER Cedex 5
Tél : 04 67 04 63 00 - Fax : 04 67 63 57 95

olivier.naud@montpellier.cemagref.fr
www.montpellier.cemagref.fr/umritap

12 juin 2007

Table des matières

1	Introduction	3
1.1	Description d'une exploitation	3
1.2	Méthodologie envisagée	3
1.3	Outils de modélisation envisagés	4
2	Modélisation du système viticole étudié	5
2.1	Les Systèmes temps réel	5
2.2	Outils de modélisation du Système	6
2.2.1	Les automates temporisés	6
2.2.2	L'automate des régions (graphe des régions)	8
2.3	Outils de modélisation des requêtes de vérification	11
2.3.1	La logique CTL	11
2.3.2	La logique TCTL	15
2.4	Outils de vérification	16
2.4.1	Model-checker UPPAAL	16
2.4.2	Model-checker KRONOS	17
2.5	Conclusion	17
3	Model-checking pour prévoir une action à réaliser	19
3.1	introduction	19
3.2	Model-checking CTL des automates temporisés	19
3.2.1	Algorithme de model-checking CTL	20
3.3	Model-checking TCTL des automates temporisés	21
3.4	Conclusion	22
4	Méthodologie "décision - Ordonnancement"	23
4.1	Description détaillée de la problématique	23
4.1.1	Concepts à modéliser	23
4.1.2	Décisions et fenêtres temporelles de mise en œuvre	25
5	Ordonnancement	29
5.1	Problème et notations	29
5.2	Heuristique employée pour la confection de lots et notations	30
5.3	Programmation linéaire en nombres entiers	31
5.3.1	Programme linéaire en nombres entiers	31
5.3.2	Algorithme d'approximation	33
5.4	Programmation par contraintes	33
5.4.1	Concepts de modélisation	34
5.4.2	La modélisation dans CHOCO	36

5.4.3	Résultats	42
6	Conclusion	50
6.1	Limites	50
6.2	Perspectives	51
6.3	Conclusion	51
	Bibliographie	52

Chapitre 1

Introduction

De nos jours le respect de l'environnement est devenu une des priorités de nos Etats et a amené au concept de production agricole durable. Les exigences de cette production conduisent à des procédés de plus en plus complexes et la création de nouvelles règles de gestion devient nécessaire au bon fonctionnement de ces productions. Ceci est spécialement important pour le traitement des maladies de la vigne qui nécessite de nombreux traitements chimiques. Dans cette optique, des recherches sur les traitements phytosanitaires des vignes ont abouti à un ensemble de règles de décision permettant de réduire le nombre de traitements chimiques à effectuer par les exploitants. Mais l'application de ces règles peut avoir un impact significatif sur l'organisation du travail. La mise en place de ces règles doit donc satisfaire l'environnement mais également les viticulteurs. Nous devons donc mettre en place un processus de décision permettant de répondre à ces critères.

L'objectif de ce stage est d'élaborer une méthodologie d'évaluation de l'applicabilité temporelle de règles de décision, et des processus de mise en œuvre associés. Le travail de ce mémoire s'inscrit dans le cadre du projet "Vin et Environnement" débuté en octobre 2005. Il fait suite à un premier travail effectué par Tuitete Tu durant son stage de Master en 2006 [Tui06].

1.1 Description d'une exploitation

Une exploitation viticole peut être décrite par le nombre de parcelles qui la composent et leurs surfaces, le nombre de tracteurs dont elle dispose et ses ressources humaines. Chacun de ces aspects est lui aussi caractérisé par différents attributs. Pour les parcelles nous pouvons les distinguer par leurs superficies, les tâches à effectuer dessus, leur stade phénologique (stade de croissance de la vigne) et le niveau d'infestation par les pathologies (maladies) considérées. Les tracteurs seront caractérisés par la capacité de leur cuve et leur vitesse de traitement. Les ressources humaines seront caractérisées par le nombre d'heures de travail possibles sur une journée, ainsi que le nombre de tâches pouvant être effectuées. Le stade phénologique d'une parcelle évolue suivant le temps et suivant le cépage. La pathologie d'une parcelle évolue suivant le temps, le climat et les traitements phytosanitaires. De ce fait, nous nous devons de considérer que les parcelles évoluent indépendamment les unes des autres. A partir de ces observations nous pouvons considérer l'exploitation comme un système réactif évoluant avec le temps.

1.2 Méthodologie envisagée

Nous souhaitons créer un processus de décision jusqu'à la mise en œuvre des actions décidées. Nous proposons d'effectuer une succession d'étapes de vérification et de résolution. Pour cela, nous

allons modéliser notre système sous forme d'automate temporisé (chapitre 2 section 2.2.1). Nous aurons un automate pour chacune des parcelles de l'exploitation qui définira son stade phénologique et un autre pour son stade pathologique. Nous aurons également un automate pour modéliser l'évolution climatique (pluies) et un automate pour chacune des règles de gestion. Nous modéliserons des requêtes temporelles sous forme de formules de la logique temporelle TCTL (chapitre 2 section 2.3.2), ces dernières nous permettront de vérifier grâce au model-checking des propriétés spécifiques au système étudié. Par exemple : si j'ai traité à la date D est ce que ma vigne sera toujours protégée si une pluie se présente à la date T ?(vivacité bornée)

Grâce au model-checking (vérification) nous avons la possibilité d'affecter des fenêtres temporelles aux tâches à effectuer sur nos parcelles. Une fois ces fenêtres temporelles calculées, nous créons un ordonnancement (chapitre 5). Cet ordonnancement ayant une forte combinatoire, dû au nombre important d'automates, nous mettons en place une heuristique de confection de lot qui permet de découper notre problème d'ordonnancement.

1.3 Outils de modélisation envisagés

Lors de ce mémoire nous décrirons plus précisément les outils suivants :

L'automate temporisé (chapitre 2 section 2.2.1) est un automate fini auquel on ajoute une représentation du temps continu sous la forme d'horloges. Ces horloges permettent d'introduire des contraintes temporelles qui peuvent être associées aux états et aux transitions de l'automate discret. Les automates temporisés nous permettront donc de modéliser formellement des états, des dates et des durées. Dans notre cas, les automates temporisés nous permettront de modéliser nos exploitations ainsi que l'évolution de leurs besoins par rapport au temps et autres événements, ainsi que les règles de gestion que l'on souhaite mettre en place.

La logique TCTL (chapitre 2 section 2.3.2) est une logique temporelle permettant de représenter les propriétés des systèmes à temps réel, notamment la " vivacité bornée ". La propriété de cette logique temporelle est qu'elle représente du temps dit "dense" (dans \mathbb{R}). De ce fait, entre un présent et un futur il y a une multitude de séquences possibles.

Le model-checking (chapitre 3) est un ensemble de techniques automatisées permettant de vérifier les propriétés des systèmes à temps réels qui sont : l'atteignabilité, la vivacité, la vivacité bornée, la sûreté et l'absence de blocage. Dans le cas de la TCTL et des automates temporisés, le model-checking se fait par la création d'un " graphe des régions " permettant de discrétiser le temps suivant les contraintes associées aux horloges. Le model-checking est utilisé dans la vérification de programmes multi-tâches ou de communication réseau et également dans des systèmes industriels tel que le contrôle ferroviaire.

Puis nous aborderons **le problème d'ordonnancement et l'heuristique de confection de lots (chapitre 5)**. Mais dans un premier temps nous allons décrire **les propriétés des Systèmes temps réel**.

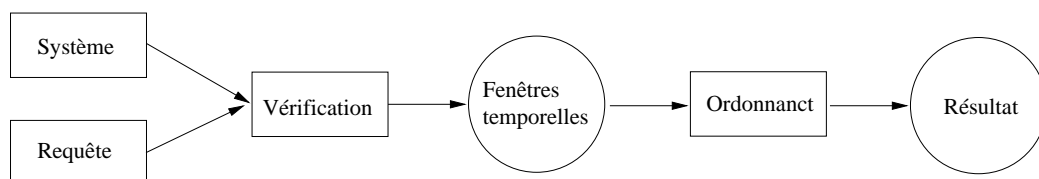


FIG. 1.1 – Méthode de résolution qui sera affinée par la suite

Chapitre 2

Modélisation du système viticole étudié

2.1 Les Systèmes temps réel

Nous distinguons dans la littérature plusieurs types de propriétés. Nous présentons dans ce paragraphe une classification des propriétés temporelles souvent rencontrées dans les travaux de spécification des systèmes temps réel [Cha03].

– Propriétés d'exactitude

Le système fait au moins ce pourquoi il a été conçu.

– Propriétés de sûreté

Une propriété de sûreté énonce que, sous certaines conditions, un évènement et/ou une situation ne se produit jamais.

Exemples :

(S1) " Les deux systèmes ne seront jamais simultanément en section critique "

(S2) " Il n'y aura jamais de débordement mémoire "

– Propriétés de vivacité

Nous distinguons dans la littérature plusieurs formes de vivacité, la vivacité simple et la vivacité bornée.

– Propriétés de vivacité simple

Sous certaines conditions, un évènement et/ou une situation finira par avoir lieu.

Exemples :

(V1) " Une requête est satisfaite un jour "

(V2) " Le système peut toujours retourner à l'état initial "

– Propriétés de vivacité bornée

Sous certaines conditions, quelque chose finira par avoir lieu en un temps borné.

Exemples :

(VB1) " Toute requête finira par être satisfaite en au moins de 5 mn "

(VB2) " Le feu passera au vert en 3mn au plus "

2.2 Outils de modélisation du Système

2.2.1 Les automates temporisés

Le formalisme des automates temporisés [AD90] étend celui des automates à états finis en rajoutant des variables réelles positives exprimant des horloges. Les définitions de cette section sont tirées pour l'essentiel de la thèse de S.Yovine [Yov93].

Définition : Soit X un ensemble d'horloges à valeur dans \mathbb{R} . L'ensemble $\Phi(X)$ des contraintes temporelles sur X est défini par la grammaire suivante :

$$\phi := true \mid x \prec c \mid x - y \prec c \mid \neg\phi \mid \phi \wedge \phi$$

avec :

- x et y des horloges,
- $c \in \mathbb{Q}^+$,
- $\prec \in \{=, <, \leq, \geq, >\}$,
- $x - y$: une opération arithmétique sur des horloges.

La disjonction entre deux contraintes temporelles peut-être définie par :

$$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$$

Définition : Un automate temporisé est représenté par un graphe G qui est un n -uplet (S, X, E, A, Inv) avec :

- S : un ensemble fini de sommets avec s_0 le sommet initial,
- X un ensemble fini d'horloges,
- E un ensemble fini d'étiquettes ou de labels,
- A un sous-ensemble fini du produit cartésien $S \times E \times \Phi(X) \times 2^X \times S$ définissant les arcs du graphe. Un arc sera donc un n -uplet (s, e, ϕ, δ, s') où s et s' seront respectivement les sommets source et destination de l'arc, e un label, $\phi \in \Phi(X)$ sera une contrainte temporelle sur un sous-ensemble des horloges de X appelée **garde** et δ un sous-ensemble d'horloges de X à réinitialiser à 0.
- $Inv(s) : S \rightarrow \Phi(X)$ une application qui associe à chaque sommet de S une contrainte temporelle appelée **invariant** du sommet.

Remarque : Les valeurs des horloges croissent uniformément en fonction du temps, c'est-à-dire que si la valeur d'une horloge augmente de δ_t , alors toutes les horloges en font de même.

Avec ce formalisme, les contraintes temporelles peuvent être exprimées sur les arcs ainsi que sur les sommets du graphe. Le système pourra franchir un arc seulement si les contraintes temporelles de l'arc exprimées sur les horloges sont satisfaites. De même, le système ne pourra rester dans un état que s'il satisfait l'invariant du sommet. De plus, on impose que le franchissement des arcs se fasse instantanément, c'est à dire que le temps ne s'écoule que dans les états de l'automate et pas sur les arcs. On notera que la définition d'Alur [AD90] ne contient pas, elle, d'invariant sur les sommets.

Définition : Soient $G_1 = (S_1, X_1, E_1, A_1, Inv_1)$ et $G_2 = (S_2, X_2, E_2, A_2, Inv_2)$ deux automates. Le produit $G = G_1 || G_2$, en imposant $X_1 \cap X_2 = \emptyset$, est l'automate (S, X, E, A, Inv) avec :

- Les sommets résultent du produit cartésien de S_1 et S_2 i.e $S = S_1 \times S_2$
- L'ensemble X des horloges est l'union des ensembles d'horloges X_1 et X_2 i.e $X = X_1 \cup X_2$.
- L'ensemble E des étiquettes est l'union des ensembles d'étiquettes E_1 et E_2 i.e $E = E_1 \cup E_2$.
- L'invariant du sommet $(s_1, s_2) \in S_1 \times S_2$ est constitué de la conjonction des contraintes temporelles Inv_1 et Inv_2 i.e $Inv(s_1, s_2) = Inv_1(s_1) \wedge Inv_2(s_2)$.

- Avec $a_1 = (s_1, e_1, \phi_1, \delta_1, s'_1) \in A_1$ et $a_2 = (s_2, e_2, \phi_2, \delta_1, s'_2) \in A_2$, l'ensemble des arcs de G sont obtenus ainsi :
 - si $e_1 = e_2$, alors $((s_1, s_2), e_1, \phi_1 \wedge \phi_2, \delta_1 \cup \delta_2, (s'_1, s'_2)) \in A$,
 - si $e_1 \neq e_2$, alors $((s_1, s_2), e_1, \phi_1 \wedge \phi_2, \delta_1 \cup \delta_2, (s'_1, s_2)) \in A$ ainsi que $((s_1, s_2), e_2, \phi_1 \wedge \phi_2, \delta_1 \cup \delta_2, (s_1, s'_2)) \in A$.

Définition : Une valuation v est une fonction qui associe à toute horloge de X une valeur dans \mathbb{R}^+ , autrement dit $v(X) \in \mathbb{R}^+$. L'ensemble des valuations est noté V .

Pour $t \in \mathbb{R}^+$, $v + t$ est l'interprétation d'horloge v' définie par $v'(X) = v(X) + t$.

Soit $h \subseteq X$, on définit enfin $v[h := 0]$, la valuation des horloges de X après remise à zéro des horloges appartenant à h , par :

$$v[h := 0](x) = \begin{cases} 0 & \text{si } x \in h \\ v(x) & \text{sinon} \end{cases}$$

Contrairement à un automate à états finis où un état est déterminé par un sommet de son graphe, l'état d'un automate temporisé sera déterminé par un couple $q = (s, v)$, $q \in Q$ correspondant à un sommet de $S \in G$ associé à une valuation v des horloges de X sur ce sommet.

Définition : Une transition dans un automate temporisé représente le franchissement d'un arc ou la progression du temps dans un même sommet :

- **Transition discrète :** Soit l'arc (s, e, ϕ, δ, s') . L'état $q = (s, v)$ a une transition discrète vers $q' = (s', v')$ si v satisfait la contrainte ϕ et $v' = v[\delta := 0]$ permet de prendre en compte la réinitialisation des horloges de δ . On note cette transition $(s, v) \xrightarrow{e} (s', v')$.
- **Transition temporelle :** Soit $t \in \mathbb{R}^+$. L'état (s, v) a une transition temporelle vers $(s, v + t)$ notée $(s, v) \xrightarrow{t} (s, v + t)$ si, pour tout $t' \leq t$, $v + t'$ satisfait $Inv(s)$.

Définition : Le triplet $(Q, q_0, \longrightarrow)$ est le **système de transition** de l'automate temporisé avec $q_0 \in Q$ un ensemble d'états initiaux. Ce système de transition définit ce que l'on appelle le modèle $M(G)$ de l'automate temporisé G .

Définition : Une séquence est une succession infinie d'états et de transitions $q_0 \xrightarrow{1} q_1 \xrightarrow{2} q_2 \xrightarrow{3} q_3 \dots q_{n-1} \xrightarrow{n} q_n$ avec $\xrightarrow{i} \in \longrightarrow$.

Remarque : L'ensemble de toutes les séquences possibles va représenter finalement le modèle $M(G)$ de l'automate temporisé de manière extensive. Précisons que le nombre d'états de ces séquences est potentiellement infini puisque que le temps est considéré comme dense c'est-à-dire qu'entre deux dates t_1 et t_2 il existe toujours une date t' .

Définition : Un automate temporisé est dit Zénon s'il est dans l'une des situations suivantes :

- il peut conduire à un état où le temps ne peut plus évoluer et aucune action ne peut être produite.
- il peut effectuer un nombre infini d'actions discrètes dans un intervalle fini de temps.

2.2.2 L'automate des régions (graphe des régions)

L'automate des régions est utilisé pour vérifier toutes les séquences possibles d'un automate temporisé. Nous verrons par la suite que cette vérification peut se ramener, grâce à l'automate des régions, en un model-checking de type CTL. Nous venons de voir qu'une séquence a un nombre infini d'états du fait que le temps est considéré comme dense. L'automate des régions sert à discrétiser ce temps en introduisant la notion de région temporelle. Dans un premier temps nous définirons cette notion puis nous étudierons la construction de cet automate. Les définitions de cette section sont tirées d'Alur [AD94].

Nous introduisons les notations suivantes :

c_x : la plus grande constante à laquelle l'horloge x est comparée dans une contrainte d'horloge de l'automate.

Pour tous $t \in \mathbb{R}$:

$\lfloor t \rfloor$: partie entière de t .

$\text{frac}(t)$: partie fractionnaire de t .

Définition : Equivalence d'horloges " \sim " : Soient v et v' deux valuations. $v \sim v'$ si et seulement si $x \in X$ et les trois conditions suivantes sont vérifiées :

- Pour toute horloge $x \in X$ on a $v(x) > c_x$ et $v'(x) > c_x$, ou $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$.
- Pour toute horloge $x \in X$ avec $v(x) \leq c_x$, $\text{frac}(v(x)) = 0$ ssi $\text{frac}(v'(x)) = 0$.
- pour tout couple d'horloges $x_i, x_j \in X$ avec $v(x_i) \leq c_{x_i}$ et $v(x_j) \leq c_{x_j}$, $\text{frac}(v(x_i)) \leq \text{frac}(v(x_j))$ ssi $\text{frac}(v'(x_i)) \leq \text{frac}(v'(x_j))$.

On note $[v]$ la classe d'équivalence (**région**) à laquelle appartient v .

Lemme : Tant que l'on reste à l'intérieur d'une même région, la valeur des contraintes d'horloge ne peut pas varier.

Exemple :

Soit A un automate contenant 2 horloges x et y , avec $c_x = 2$ et $c_y = 1$. Les contraintes sur ses horloges sont :

- sur x : $\{x = c \mid c = 0, 1, 2\} \cup \{c - 1 < x < c \mid c = 1, 2\} \cup \{x > 2\}$
- sur y : $\{y = c \mid c = 0, 1\} \cup \{0 < y < 1\} \cup \{y > 1\}$

Nous obtenons **28 régions d'horloge** :

- 6 points d'angle. Ex : $x = 1$ et $y = 0$
- 14 segments ouverts. Ex : $x = 1$ et $0 < y < 1$
- 8 surfaces ouvertes. Ex : $1 < x < 2$ et $1 < y$

Sur la figure 2.1, le premier point de la définition correspond au tracé des droites parallèles aux axes des abscisses et des droites parallèles aux axes des ordonnées.

Le deuxième point de la définition indique que les droites qui viennent d'être tracées constituent elles-mêmes des régions. D'une manière générale, une droite participe à la formation de plusieurs régions.

Le troisième et dernier point de la définition se traduit par le tracé des droites parallèles à la première bissectrice.

Pour exprimer l'évolution du temps dans l'automate des régions, nous devons définir des notions de successeur temporel et de successeur temporel direct appliquées aux régions.

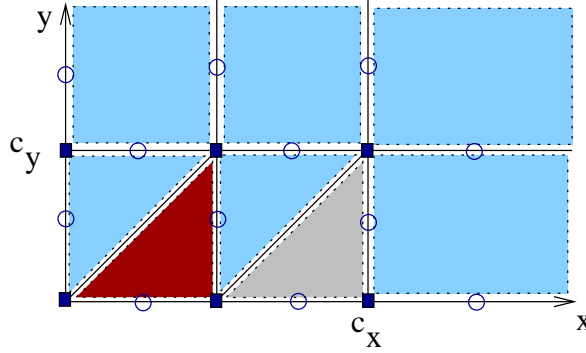


FIG. 2.1 – Représentation des 28 régions

Définition : Successeur d'une région : Soient α et α' deux régions, α' est successeur de α si et seulement si :

$$\forall v \in \alpha, \quad \exists t \in \mathbb{R}^+, \quad | \quad v+t \in \alpha'$$

Les successeurs de la région $\alpha = (0 < y < x < 1)$ (en rouge sur la figure 2.1) sont :

- $0 < y < 1$ et $x = 1$
- $0 < y < 1$ et $1 < x < y+1$
- $y = 1$ et $1 < x < 2$
- $1 < y$ et $1 < x < 2$
- $1 < y$ et $x = 2$
- $1 < y$ et $2 < x$

Ces régions correspondent à toutes les régions qui peuvent être rencontrées en traçant une droite parallèle à l'axe $x = y$, et en se déplaçant dans le sens croissant des valeurs de x et y , depuis un point quelconque appartenant α .

Définition : Successeur **direct** d'une région : Une région d'horloges α' est le successeur temporel direct d'une région d'horloges α , si α' est un successeur temporel de la région α et s'il n'existe pas de région $\alpha'' \neq \alpha'$ telle que α'' est le successeur temporel de α et α' est le successeur temporel de α'' .

Définition : L'automate des régions : L'automate des régions $\mathcal{R}(A)$ de l'automate temporisé $A = (S, X, E, A, Inv)$ est un automate défini sur l'alphabet E ;

- Les états de $\mathcal{R}(A)$ sont de la forme $\langle s, \alpha \rangle$, où $s \in S$ et α est une région d'horloge,
- Les états initiaux de $\mathcal{R}(A)$ sont de la forme $\langle s_0, [v_0] \rangle$, où $s_0 \in S$ est le sommet initial de A et $v_0(x) = 0 \quad \forall x \in X$,
- Il y a un arc $(\langle s, \alpha \rangle, \langle s', \alpha' \rangle, a)$ dans $\mathcal{R}(A)$ si et seulement si il existe une transition (s, e, ϕ, δ, s') dans A et une région d'horloge α'' telle que α'' soit un successeur temporel de α , que α'' satisfasse ϕ , et que $\alpha' = [\delta := 0]\alpha''$.

Remarque : les états de cet automate des régions ne font intervenir par construction que les transitions discrètes de l'automate temporisé ($e \in E$) et la notion de région successeur. La notion de successeur direct n'est pas mobilisée.

Exemple :

Nous allons détailler la construction d'un automate des régions, la base de cet exemple est tiré de Alur [AD94]. Soit l'automate temporisé A suivant :

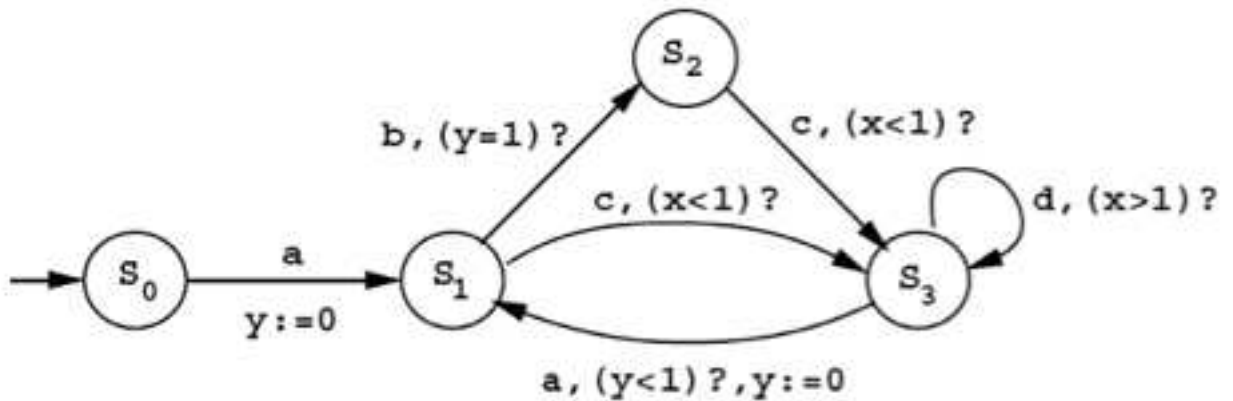
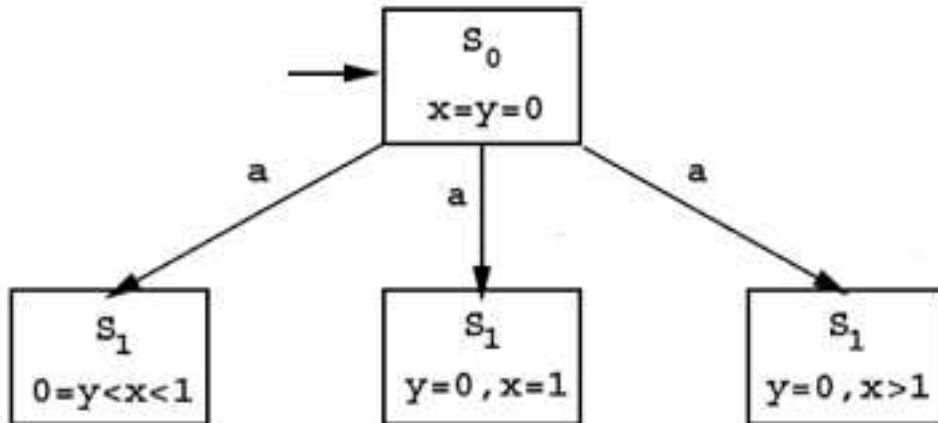


FIG. 2.2 – Automate temporisé A

Nous avons $c_x = 1$ et $c_y = 1$. De l'état initial s_0 on n'atteint le sommet s_1 que par l'action a et l'horloge y est réinitialisé (la valeur de l'horloge x est quelconque). On obtient trois régions successeur de la région $\alpha\{x = y = 0\}$ qui sont les régions définies par $\alpha'_1\{0 < y = x < 1\}$, $\alpha'_2\{x = y = 1\}$ et $\alpha'_3\{1 < x = y\}$ et qui satisfont $\phi = true$. Nous avons donc trois régions $\alpha'_1\{y = 0 < x < 1\}$, $\alpha'_2\{y = 0, x = 1\}$, $\alpha'_3\{y = 0, 1 < x\}$ issues respectivement de $[y := 0]\alpha'_1$, $[y := 0]\alpha'_2$ et $[y := 0]\alpha'_3$.

Depuis l'état s_0 de A nous pouvons atteindre l'état s_1 par l'action a mais dans trois régions distinctes. Nous obtenons le "début" d'automate suivant :



Nous obtenons l'automate de la figure 2.3.

Remarque : Les relations temporelles inscrites dans chacun des états de l'automate des régions sont vérifiées par les horloges quand l'automate atteint cet état. Ces relations ne sont pas assimilables aux invariants définis pour les états discrets des automates temporisés puisque l'automate peut rester dans cet état discret et voir la valeur de ses horloges évoluer pour atteindre une nouvelle région.

Remarque : La satisfaction d'une formule TCTL par un automate temporisé peut être établie sur l'abstraction fini de l'automate, qu'est l'automate des régions. N.Halbwachs. (tiré de la thèse de A.Gouin [Gou99])

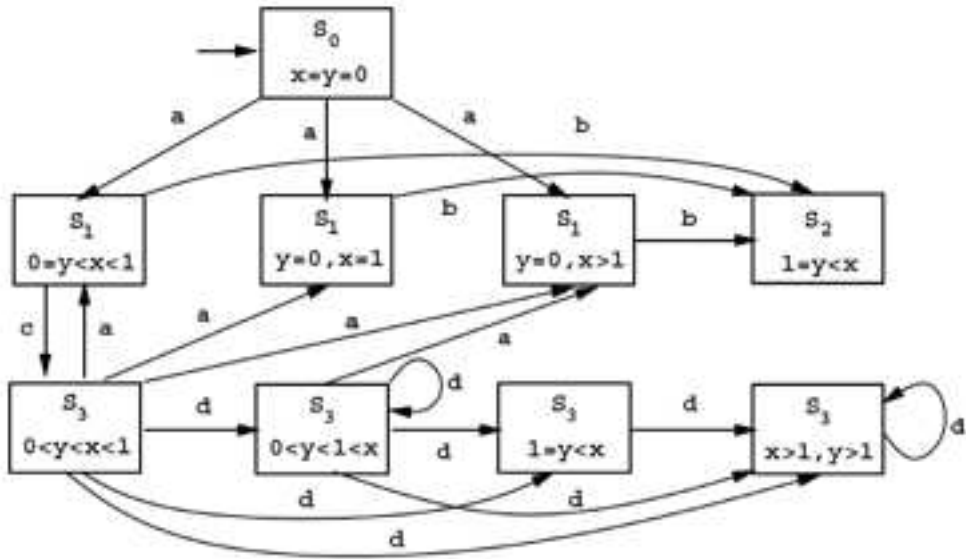


FIG. 2.3 – Automate des régions de l'automate temporisé A

2.3 Outils de modélisation des requêtes de vérification

Pour modéliser nos requêtes de vérification, nous allons utiliser la logique temporelle. La logique temporelle est une extension de la logique modale, elle intègre de nouveaux opérateurs qui expriment la notion de séquence et de temps. Grâce à cette notion de temps, nous pouvons exprimer des actions futures et passées, et donc vérifier le comportement d'un système tout au long de son évolution. Nous étudierons dans cette section, la logique CTL 2.3.1 et la logique TCTL 2.3.2.

Les définitions suivantes sont tirées du mémoire de stage de Master de Tuitete Tu [Tui06]

Définition : Soient PA un ensemble de propositions logiques atomiques et $P : S \rightarrow 2^{PA}$ une application qui à tout sommet $s \in S$ associe un ensemble de propositions atomiques $P(s)$ de PA .

2.3.1 La logique CTL

La CTL (Computational Tree Logic), introduite au début des années 80 par Clarke et Emerson, permet de considérer des modèles de systèmes dans lesquels il existe plusieurs séquences d'exécutions finies à partir d'un état donné du système, d'où la notion d'arbre. Cette logique s'oppose à la logique "Propositional Linear Temporal Logic" (PLTL) où l'on ne considère à chaque fois qu'une seule séquence finie d'état du système vérifiant une propriété donnée.

Syntaxe CTL

Les formules CTL sont définies par la grammaire suivante :

$$\phi := p \mid \neg\phi \mid \phi \vee \phi \mid \exists \bigcirc \phi \mid \exists [\phi \cup \phi] \mid \forall [\phi \cup \phi]$$

avec $p \in PA$ une proposition logique atomique, et les opérateurs suivants :

- \exists : il existe,
- \forall : quel que soit,
- \bigcirc : suivant,
- \cup : jusqu'à.

Les opérateurs usuels booléens de conjonction, d'implication et d'équivalence entre deux formules CTL ϕ et ψ peuvent être définis comme des macros par :

$$\begin{aligned}\phi \wedge \psi &\equiv \neg(\neg\phi \wedge \neg\psi) \\ \phi \Rightarrow \psi &\equiv \neg\phi \vee \psi \\ \phi \iff \psi &\equiv \phi \Rightarrow \psi \wedge \psi \Rightarrow \phi\end{aligned}$$

Modèle CTL

Définition : Un modèle CTL est ce que l'on appelle une structure de Kripke $M = (S, P, R)$ ¹ où :

- S est un ensemble non vide d'états, $S_0 \subseteq S$ est l'ensemble des états initiaux,
- $P : S \longrightarrow 2^{PA}$ une application qui à tout sommet $s \in S$ associe un ensemble de propositions logiques atomiques $P(s)$ de PA .
- $R \subseteq S \times S$ est une relation totale sur S , c'est à dire que pour chaque état $s \in S$ les états suivants possibles sont donnés par R .

Définition : Une séquence est une succession infinies d'états (s_0, s_1, \dots) tel que $\forall i, (s_i, s_{i+1}) \in R$.

Sémantique CTL

La sémantique CTL est définie par une relation de satisfaction notée \models et relative à un modèle M . On a $(M, s) \models \phi$ si et seulement si ϕ est satisfaite en s . On note souvent plus simplement $s \models \phi$ lorsque M est implicite. Cette relation est définie par induction de la manière suivante :

- $s \models p$ si et seulement si $p \in P(s)$ i.e la proposition p est associée à s .
- $s \models \neg\phi$ si et seulement si $\neg(s \models \phi)$ i.e s'il est faux que le sommet s puisse vérifier la formule ϕ .
- $s \models \phi_1 \vee \phi_2$ si et seulement si $(s \models \phi_1) \vee (s \models \phi_2)$
- $s \models \exists \bigcirc \phi$ si et seulement si $(s, t) \in R$ et $t \models \phi$ i.e si ϕ est vraie pour au moins un successeur de s .
- $s \models \exists \phi_1 \cup \phi_2$ si et seulement si il existe au moins une séquence (s_0, s_1, \dots) avec $s = s_0$ tel que quelque soit (i, j) , $0 \leq i \leq j$, $s_i \models \phi_1$ et $s_j \models \phi_2$. Autrement dit, s'il existe au moins une séquence vérifiant ϕ_1 en s_0 et ϕ_2 plus loin dans la séquence.
- $s \models \forall \phi_1 \cup \phi_2$ si et seulement si pour toutes les séquences (s_0, s_1, \dots) avec $s = s_0$ tel que quelque soit (i, j) , $0 \leq i \leq j$, $s_i \models \phi_1$ et $s_j \models \phi_2$. Autrement dit, si toutes les séquences vérifient ϕ_1 en s_0 et ϕ_2 plus loin dans la séquence.

Enfin on note :

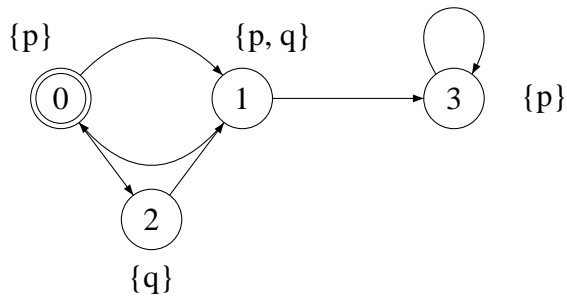
- $\exists \diamond \phi \equiv \exists \text{vrai} \cup \phi$ qui dit qu'il existe une séquence où la formule ϕ est satisfaite au moins une fois.

¹La logique CTL fait partie des logiques modales.

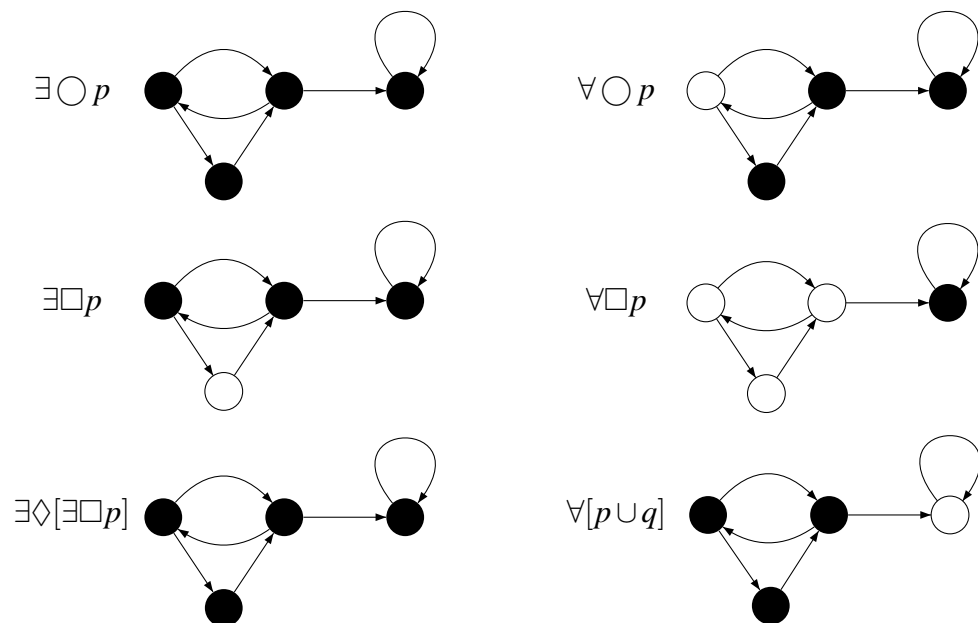
- $\exists \Box \phi \equiv \exists \neg \Diamond \neg \phi$ qui dit qu'il existe une séquence où la formule ϕ est toujours vraie (ie ϕ est un invariant).

Exemple :

L'exemple suivant est tiré de [Kat98]. Soit le modèle M représenté par la structure de Kripke suivante :



Dans les figures suivantes nous avons coloré en noir les sommets vérifiant une formule CTL particulière.



- $\exists \bigcirc p$ est valide sur tous les états puisqu'ils ont tous au moins un successeur vérifiant p .
- $\forall \bigcirc p$ est valide sur les états 1, 2 et 3 puisque leurs successeurs vérifient tous p alors que l'état 0 possède 1 comme successeur ne vérifiant pas p .
- $\exists \square p$ n'est pas valide en 2 puisque l'état 2 ne vérifie pas la proposition p alors que sur les autres états, il existe à chaque fois une séquence sur laquelle la proposition p est toujours valide.
- $\forall \square p$ n'est valide qu'en 3 puisque toutes les séquences commençant dans cet état valident p . Autrement dit, p est un invariant à partir de l'état 3.
- $\exists \diamond [\exists \square p]$ est valide dans tous les états puisqu'il existe toujours dans ces états une séquence menant à un état dans lequel il existe une séquence validant toujours p .
- $\forall [p \vee q]$ n'est pas valide en 3 puisque toutes les séquences issues de l'état 3 ne valident pas q .

Expressivité (tiré de [Cha03])

Propriété	Code	Formule
Sûreté	S1	$\forall \square \neg (SC1 \wedge SC2)$
	S2	$\forall \square \neg Dbordement$
Vivacité	V1	$\forall \square (req \Rightarrow \exists \diamond sat)$
	V2	$\forall \square init$
Absence de blocage	PB	$\forall \square \exists \bigcirc vrai$

2.3.2 La logique TCTL

La "Timed Computation Tree Logic" étend la CTL en intégrant une notion de temps borné. Ainsi des formules peuvent par exemple être vérifiées dans un état pendant un certain temps t borné avant qu'elles ne soient invalidées. La logique TCTL permet, grâce à cette notion, de vérifier des propriétés des systèmes à temps réel où les séquences d'exécutions ne sont pas finies. Cette logique est très souvent associée aux automates temporisés.

Syntaxe TCTL

Les formules TCTL sont définies par la grammaire suivante :

$$\phi := p \mid x \prec c \mid x - y \prec c \mid \neg \phi \mid \phi \vee \phi \mid \exists [\phi \cup_{\prec c} \phi] \mid \forall [\phi \cup_{\prec c} \phi]$$

avec x et y des horloges, $c \in \mathbb{Z}^+$ et $\prec \in \{=, <, \leq, \geq, >\}$.

Sémantique TCTL

Pour le modèle $M(G)$ du système de transition $(Q, q_0, \longrightarrow)$ de l'automate temporisé G , on définit la relation de satisfaction d'une formule TCTL en un état $q = (s, v)$, $q \in Q$ par induction :

- $q \models p$ si et seulement si $p \in P(s)$ i.e la proposition p est associée à s .
- $q \models x \prec c$ si et seulement si $v(x) \prec c$ i.e la valuation de x respecte cette contrainte d'horloge.
- $q \models x - y \prec c$ si et seulement si $v(x) - v(y) \prec c$ i.e la valuation de x moins celle de y respecte cette contrainte d'horloge.
- $q \models \neg \phi$ si et seulement si $\neg (s \models \phi)$ i.e s'il est faux que le sommet s puisse vérifier la formule ϕ .
- $q \models \phi_1 \vee \phi_2$ si et seulement si $(s \models \phi_1) \vee (s \models \phi_2)$
- $q \models \exists \phi_1 \cup_{\prec c} \phi_2$ si et seulement si il existe au moins une séquence (q_0, q_1, \dots) avec $q = q_0$ tel que quelque soit (i, j) , $0 \leq i \leq j$, $q_i \models \phi_1$ et $q_j \models \phi_2$ avec un temps entre q_0 et q_j satisfaisant la relation $\prec c$. Autrement dit, s'il existe une séquence où ϕ_1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte $\prec c$ où ϕ_2 est vraie.
- $q \models \forall \phi_1 \cup_{\prec c} \phi_2$ si et seulement si pour toutes les séquences (q_0, q_1, \dots) avec $q = q_0$ tel que quelque soit (i, j) , $0 \leq i \leq j$, $q_i \models \phi_1$ et $q_j \models \phi_2$ avec un temps entre q_0 et q_j satisfaisant la relation $\prec c$. Autrement dit, si pour toutes les séquences ϕ_1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte $\prec c$ où ϕ_2 est vraie.

Remarque : Ici, l'état q d'un automate temporisé est un couple (s, v) alors que dans la logique CTL l'état d'un automate se réduisait à un sommet de son graphe.

Remarque : L'opérateur \circ (suivant) n'existe pas en TCTL. En effet, cette logique travaillant sur du temps borné et réel, il peut y avoir une infinité d'état entre deux états d'une séquence. De ce fait la notion de suivant n'a plus de sens.

Expressivité (tiré de [Cha03])

Propriété	Code	Formule
Sûreté	S1	$\forall \square \neg (SC1 \wedge SC2)$
	S2	$\forall \square \neg Dbordement$
Vivacité	V1	$\forall \square (req \Rightarrow \exists \diamond sat)$
	V2	$\forall \square init$
Vivacité Borné	VB1	$\forall \square (req \Rightarrow \exists \diamond_{<5} sat)$
	VB2	$\forall \square \exists \diamond_{<3} vert$

2.4 Outils de vérification

Nous présentons dans cette section deux outils de model-checking UPPAAL [GB02] et KRONOS [CD95]

2.4.1 Model-checker UPPAAL

UPPAAL a été créé par l'université Uppsala en Suède et l'université Aalborg au Danemark. Les fondements théoriques ont été publiés en 1994 par W.Yi, P.Pettersson et M.Daniels [WY94]. La première version a été distribuée en 1995.

UPPAAL est un model-checker intégrant une interface graphique permettant la construction d'automates temporisés. Lors de cette construction nous pouvons spécifier pour un état, en plus de ses invariants, si il est "urgent" ou "committed".

Urgent Un état de contrôle est dit "urgent" lorsqu'on ne peut attendre dans aucun états du système, lorsque le système atteint cet état. *Le système est donc obligé de faire des transitions instantanées, d'un état du système à un autre, tant que le système est dans un état urgent.*

Committed Un état de contrôle est dit "committed" lorsqu'on ne peut pas attendre dans cet état(= urgent) et quand on est dans un tel état, la prochaine transition doit faire sortir de l'état (ou d'un états) committed.

Nous pouvons également définir les horloges locales (propre à un automate) ou globales (communes à tous les automates). UPPAAL permet de vérifier les fomules suivantes :

$\exists \diamond p$: il existe un chemin le long duquel p est vrai un jour.

$\exists \square p$: il existe un chemin le long duquel p est toujours vrai.

$\forall \diamond p$: le long de tout chemin p est vrai un jour.

$\forall \square p$: le long de tout chemin p est toujours vrai.

$p \Rightarrow q$: quand p est vrai, alors q sera forcément vrai un jour.

Où p et q sont du type :

$p, q := \text{état d'automate} \mid \text{horloge } \tilde{\text{valeur}} \mid p \wedge q \mid p \cup q \mid \neg p \mid p \Rightarrow q \mid \text{deadlock}$

On ne peut pas imbriquer les opérateurs. On ne peut donc pas vérifier tout TCTL.

2.4.2 Model-checker KRONOS

KRONOS est un outil développé au laboratoire VERIMAG², comme un outil de vérification par la technique de Model-Checking symbolique [Yov93] et un environnement offrant à l'utilisateur la possibilité d'assister la vérification. KRONOS permet de vérifier si un système temps réel modélisé par un automate temporisé A , sous forme textuelle, satisfait une propriété temporisée décrite par des formules TCTL. A partir d'un système à plusieurs automates $A_1 \dots A_n$ composants, KRONOS génère l'automate A qui illustre la composition globale du système.

KRONOS permet de vérifier la totalité des formules TCTL (listées en section 2.3.2), il permet donc de vérifier les propriétés d'atteignabilité, de sûreté, de vivacité et de vivacité borné d'un système à temps réel.

2.5 Conclusion

Après avoir présenté différents outils de modélisation, nous allons justifier nos choix.

Choix de l'outil de modélisation des requêtes

Notre problème est de répondre essentiellement au type de question suivant :

– Si je traite ma parcelle à telle date, sera t-elle encore protégée 15 jours après ?

Autrement dit, existe t-il une séquence s telle que si elle contient un état vérifiant la formule $\text{traiter} \wedge (\text{date} = d)$ il existe un état vérifiant la formule $\text{danger} \wedge (x - d \leq 15)$?

Nous voulons donc vérifier des propriétés de vivacité bornée sur notre système, nous choisissons donc la logique TCTL pour modéliser nos requêtes.

Choix du Model-checker

Comme nous l'avons vu dans la section 2.4 l'outil de vérification UPPAAL ne permet pas de vérifier l'ensemble des formules de la logique TCTL. Il peut néanmoins vérifier des propriétés de vivacité bornée, mais pour cela nous devons lui fournir un automate temporisé qui lui servira d'observateur. Nous préférons cependant, contrairement à Charfi[Cha03] dont nous ne partageons pas l'avis, nous tourner vers l'outil KRONOS qui permet la vérification de la totalité des formules TCTL sans aucun ajout de notre part.

Choix de l'outil de modélisation du système

D'après les travaux de F.Cassez et O.H.Roux [FC03] les réseaux de Petri temporels et les automates temporisés ont la même sémantique. De plus notre outil de vérification prend en entrée des

²<http://www-verimag.imag.fr/>

automates temporisés. Nous choisissons donc les automates temporisés comme outils de modélisation de notre système.

Chapitre 3

Model-checking pour prévoir une action à réaliser

Les définitions de ce chapitre sont tirées de Katoen [Kat03]. Tout le long de ce chapitre nous ne considérons que des automates temporisés dit *non zenon* (terme défini page 7).

3.1 introduction

Le model-checking est un ensemble de techniques automatisées permettant de vérifier les propriétés désirées des systèmes étudiés. Nous considérons dans ce travail les exploitations viticoles comme des systèmes réactifs, et nous en étudions les propriétés temporelles grâce au model-checking TCTL sur automates temporisés. Plus précisément, nous utilisons le model-checking pour décider, sur un horizon temporel de planification du travail, la nécessité de traiter pour chaque parcelle. Dans le cas où le traitement d'une parcelle donnée est nécessaire, on détermine la fenêtre temporelle au cours de laquelle le traitement doit avoir lieu. Nous verrons comment nous utilisons le model-checking TCTL (voir nos motivations chapitre 2 section 2.5) et l'outil Kronos pour réaliser cette partie de notre méthodologie, avant l'ordonnancement qui sera décrit dans le chapitre suivant.

Avant d'examiner le model-checking TCTL, nous étudierons également le model-checking CTL. En effet, comme nous le verrons dans la section 3.3, le problème du model-checking TCTL peut se ramener à un problème de model-checking CTL grâce au calcul de régions pour les valuations d'horloge.

3.2 Model-checking CTL des automates temporisés

Le model-checking CTL consiste à déterminer si une formule CTL est valide dans un modèle CTL fini $M = (S, P, R)$ tel que défini dans le chapitre 2.3 section 2.3.1. Le concept de base de l'algorithme de model-checking est de marquer tous les états $s \in S$ avec une sous-formule de ϕ qui est valide dans cet état. L'ensemble des sous-formules de ϕ sera noté $Sub(\phi)$ et est défini de la façon suivante :

Définition : Sous-formule d'une formule CTL : Soient $p \in AP$ et ϕ, ψ des formules CTL on a :

- $Sub(p) = p$
- $Sub(\neg\phi) = Sub(\phi) \cup \{\neg\phi\}$
- $Sub(\phi \vee \psi) = Sub(\phi) \cup Sub(\psi) \cup \{\phi \vee \psi\}$
- $Sub(EX\phi) = Sub(\phi) \cup \{EX\phi\}$

- $Sub(E[\phi U \psi]) = Sub(\phi) \cup Sub(\psi) \cup \{E[\phi U \psi]\}$
- $Sub(A[\phi U \psi]) = Sub(\phi) \cup Sub(\psi) \cup \{A[\phi U \psi]\}$

La procédure de marquage ci-dessus est exécutée récursivement, comme le montre l'algorithme de la section 3.2.1. Le problème de model checking pour CTL, consistant à vérifier que $M, s \models \phi$ pour un modèle CTL donné M et une formule CTL ϕ , peut maintenant être résolu pour n'importe quel état s dans S en considérant uniquement son marquage.

$$M, s \models \phi \text{ ssi } s \text{ est marqué par } \phi$$

3.2.1 Algorithme de model-checking CTL

Nous donnons ici un algorithme de base pour la résolution du problème de model-checking CTL, cet algorithme est tiré de Katoen [Kat03], il est composé de trois procédures $Sat(\phi, M)$, $Sat_{EU}(\phi_1, \phi_2, M)$ et $Sat_{AU}(\phi_1, \phi_2, M)$.

$Sat(\phi, M)$

Donné : ϕ : une formule CTL, M un model CTL.

Sortie : S un ensemble d'états.

Début

- **Si** $\phi = true \longrightarrow$ **return** S
- **Si** $\phi = false \longrightarrow$ **return** \emptyset
- **Si** $\phi \in AP \longrightarrow$ **return** $\{s \mid \phi \in Label(s)\}$
- **Si** $\phi = \neg\phi_1 \longrightarrow$ **return** $S - Sat(\phi_1, M)$
- **Si** $\phi = \phi_1 \vee \phi_2 \longrightarrow$ **return** $(Sat(\phi_1, M) \cup Sat(\phi_2, M))$
- **Si** $\phi = EX\phi_1 \longrightarrow$ **return** $\{s \in S \mid (s, s') \in R \wedge s' \in Sat(\phi_1, M)\}$
- **Si** $\phi = E[\phi_1 U \phi_2] \longrightarrow$ **return** $Sat_{EU}(\phi_1, \phi_2, M)$
- **Si** $\phi = A[\phi_1 U \phi_2] \longrightarrow$ **return** $Sat_{AU}(\phi_1, \phi_2, M)$

Fin

$$Sat(\phi, M) = \{s \in S \mid M, s \models \phi\}$$

$Sat_{EU}(\phi, \psi, M)$

Donné : ϕ, ψ : des formules CTL, M un model CTL.

Sortie : Q un ensemble d'états.

Début Q, Q' deux ensembles d'états ;

- $Q := Sat(\psi, M)$
- $Q' := \emptyset$
- **tant que** $Q \neq Q'$ **faire**
 - $Q' := Q$
 - $Q = Q \cup (\{s \mid \exists s' \in Q. (s, s') \in R\} \cap Sat(\phi, M))$
- **return** Q

Fin

$$Sat_{EU}(\phi, \psi, M) = \{s \in S \mid M, s \models E[\phi U \psi]\}$$

$Sat_{AU}(\phi, \psi, M)$

Donné : ϕ, ψ : des formules CTL, M un model CTL.

Sortie : Q un ensemble d'états.

Début Q, Q' deux ensembles d'états ;

- $Q := Sat(\psi, M)$
- $Q' := \emptyset$
- **tant que** $Q \neq Q'$ **faire**
- $Q' := Q$
- $Q = Q \cup (\{s | \forall s' \in Q. (s, s') \in R\} \cap Sat(\phi, M))$
- **return** Q

Fin

$$Sat_{AU}(\phi, \psi, M) = \{s \in S | M, s \models A[\phi U \psi]\}$$

3.3 Model-checking TCTL des automates temporisés

La relation de satisfaction \models pour TCTL n'est pas définie pour les automates temporisés car ce sont des systèmes à transition infinie dû au fait que leurs horloges sont définies sur \mathbb{R} (temps dense). Le problème du model-checking pour les automates temporisés peut être exprimé sur $M(A)$, le système de transition de l'automate A de la façon suivante. Avec $s_0 = (l_0, v_0)$ où l_0 est l'état initial de A et v_0 est la valuation d'horloge qui assigne à toutes les horloges $x \in X$ la valeur 0.

Définition : Soit ϕ une formule TCTL et A un automate temporisé on a :

$$A \models \phi \text{ ssi } M(A), (s_0, w_0) \models \phi$$

avec $w_0(x) = 0 \forall x \in X$.

Nous avons énoncé une définition de vérification sur le système de transition $M(A)$ mais les séquences de ce dernier sont encore avec un nombre d'état indénombrable, ce qui est un obstacle à la résolution concrète du problème. Nous avons vu dans le chapitre 2 section 2.2.1, que les transitions de l'automate temporisé A étaient sémantiquement équivalentes aux transitions de l'automate des régions $R(A)$ de A .

De plus le nombre d'états des transitions engendrées par l'automate des régions sont dénombrables et nous avons une notion de successeur direct qui nous permet d'avoir une notion de suivant. Nous pouvons donc réduire notre problème de model-checking sur l'automate temporisé A , à un problème de model checking sur l'automate des régions associé. L'automate des régions étant un automate à transition discrètes, le problème de model-checking peut être désormais résolu de la même façon que le problème de model-checking CTL vu en section 3.2. Nous pouvons donc dire que :

Le model checking TCTL sur un automate temporisé est équivalent au model checking CTL sur l'automate des régions correspondant.

Nous avons donc, sur le même principe que l'algorithme de model-checking CTL, un algorithme pour le model-checking TCTL.

Complexité du model checking de TCTL

Le model checking de TCTL n'est pas plus difficile que l'accessibilité ([Lar05]).

Théorème : Le model checking de TCTL sur les automates temporisés est un problème PSPACE-complet.

Le coté PSPACE-dur vient directement de la possibilité d'énoncer les propriétés d'accessibilité dans les automates temporisés. Pour montrer qu'il est dans PSPACE, il suffit de considérer un algorithme de model checking à la volée sur le graphe des régions : pour vérifier une propriété ϕ , il est suffisant de ne stocker que $|\phi|$ configurations et de procéder de manière non-déterministe.

3.4 Conclusion

Nous venons de présenter dans ce chapitre un algorithme de model-checking TCTL, et son utilité dans notre projet. Nous sommes donc à ce stade munis des outils théoriques pour calculer les fenêtres temporelles pendant lesquelles nos parcelles doivent être traitées. Nous pouvons maintenant présenter la méthodologie de résolution mise en place.

Chapitre 4

Méthodologie "décision - Ordonnancement"

Nous présentons dans ce chapitre la méthodologie adoptée pour la résolution de notre problème viticole, qui est de répondre à la question : "*Pour une exploitation donnée, correspondant à une certaine configuration, est-il possible d'implanter les règles de décisions concernant des traitements phytosanitaires à la parcelle ?*". Cette méthodologie est la même que celle utilisée par Tu Tuitete [Tui06].

4.1 Description détaillée de la problématique

Notre travail va consister dans un premier temps à construire un modèle représentant une parcelle de vigne au sein d'une exploitation. Par ailleurs, des règles de gestion concernant les traitements phytosanitaires à apporter à cette parcelle devront être intégrées dans le modèle. Ces règles prennent en compte diverses variables comme le climat, les stades phénologiques des pieds de vigne (stades de développement des plantes) ou encore le taux de maladie (champignons) de la parcelle.

Comme nous l'avons vu au chapitre 2 en modélisant une parcelle par un automate temporisé, ce dernier peut basculer d'un état à un autre en fonction d'événements. Ces événements pourront être de type temporel tel qu'une date donnée au cours des stades phénologiques de la vigne ou encore un signal de présence de maladie et donc entraînant la nécessité d'un traitement phytosanitaire.

Une exploitation pourra par la suite être modélisée comme un automate résultant du produit des automates modélisant les divers processus concepts c'est-à-dire les parcelles et les ressources matérielles et humaines de l'exploitation.

4.1.1 Concepts à modéliser

Cette partie est tirée de l'article MSR2007 [ON07]

Nous modélisons des processus de décision pour la lutte contre des maladies de la vigne ainsi que les contraintes qui s'exercent sur leur mise en œuvre. Les concepts à modéliser sont les suivants :

parcelles : Une parcelle de vigne est définie par son emplacement, sa structure physique (rangs, . . .), son accessibilité, le cépage. Au plan agronomique, on peut encore affiner cette description et le découpage spatial. Ici, la décision étant structurée par le passage d'un engin mécanisé, c'est un bloc cohérent de travail qui est considéré. Une parcelle représente, sur l'horizon de

plannification du travail, une tâche de traitement à déclencher ou non. Dans la mesure où aucun autre opérateur ne peut rentrer dans une parcelle qui est en cours de traitement ou vient d'être traitée, une parcelle est aussi une ressource, libre ou non à un instant donné.

phénologie : Les stades phénologiques codifient la pousse annuelle de la vigne depuis le débourrement (éclatement des bourgeons) jusqu'à la vendange. Bien que chaque rameau de vigne ait sa propre cinétique de développement, et que l'on rencontre donc différents stades à un moment donné sur une parcelle, on considère un stade phénologique moyen ou "dominant".

météorologie : Nous considérons températures et pluviométrie journalières. Il est bien clair que ces profils diffèrent beaucoup d'une année à l'autre. On distingue la pluie prévue par la météo, utilisée pour la décision, de la pluie réelle. Par commodité, et pour étudier l'effet des incertitudes, nous avons généré des pseudo-historiques de pluies prévues à partir d'historiques de pluies réelles, en paramétrant des erreurs de prévision. Ces erreurs concernent l'occurrence ou non d'un événement pluvieux, et un décalage dans la date prévue.

pathologies et épidémies : Nous considérons uniquement les maladies à micro-champignons, dites "cryptogamiques". Les agents pathogènes sont stockés sur les bois ou au niveau du sol pendant l'hiver. Ils se développent (mycelium) en saison sur feuilles puis sur grappes. L'épidémie se propage, à partir des "foyers primaires" qui libèrent des spores, par gravité et par le vent. L'humidité et les pluies jouent un grand rôle pour que les nouvelles contaminations soient effectives.

produits de traitement : Leur action sur le végétal et la maladie visée est caractérisée par une période de "rémanence" établie expérimentalement lors de l'homologation du produit. Cette rémanence, qui intègre les effets liés à la croissance du végétal, n'est pas à confondre avec la durée d'activité chimique du produit.

ressources matérielles : On considère, suivant la taille de l'exploitation, un ou plusieurs ensembles (tracteur+pulvérisateur). En effet, dans la mesure où les traitements phytosanitaires sont systématiquement prioritaires dans l'organisation du travail, un tracteur est toujours disponible lorsqu'un pulvérisateur l'est.

Nous nous limitons ici aux décisions concernant deux maladies cryptogamiques de la vigne, l'oïdium (*Erysiphe necator*) et le mildiou (*Plasmopara viticola*). Ces maladies, très répandues, génèrent en effet à elles seules 60% des intrants phytosanitaires viticoles. De plus, les décisions de ces interventions fongicides sont souvent associées (mêmes périodes d'application, mélanges des fongicides). Signalons enfin que l'action des fongicides est essentiellement préventive et qu'une épidémie mal maîtrisée a un fort impact sur la quantité et la qualité des raisins récoltés.

Une originalité importante des processus de décision dont nous étudions la faisabilité est que les décisions d'opportunité de traitement sont prises pour chaque parcelle. La pratique usuelle (voir figure 4.1) est en effet de réaliser les traitements sur tout ou partie d'une exploitation sur la base d'une décision unique. Ce raisonnement spécifique, qui s'accompagne d'une observation précise des symptômes des maladies, doit conduire à diminuer, globalement, le nombre de traitements.

Il reste que la mise en œuvre économique de ces "décisions à la parcelle" nécessite une coordination de l'utilisation des ressources de l'exploitation. Notre problème d'ordonnancement se pose comme dans la figure 4.2. Cependant, comme la prévision météo a un horizon limité (10 jours au grand maximum), et que la vitesse de croissance des plantes est variable suivant les années, on planifie tout au long de la saison les opérations à venir, sur un horizon d'une semaine environ (voir figure 4.3)

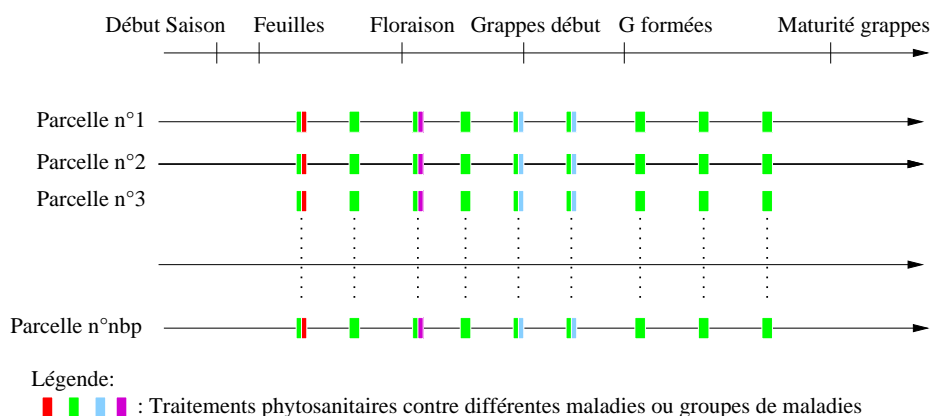


FIG. 4.1 – La pratique usuelle est de décider les traitements pour toute l'exploitation

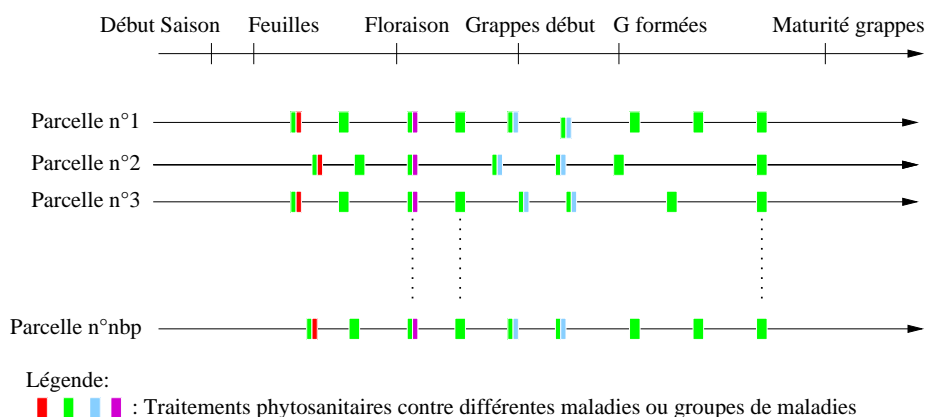


FIG. 4.2 – Décider à la parcelle et mettre en oeuvre à l'exploitation

4.1.2 Décisions et fenêtres temporelles de mise en oeuvre

On recherche, à partir des automates temporisés suivants, quelles sont les parcelles à traiter et les fenêtres temporelles où ces traitements sont possibles. Le model-checking est mis en oeuvre sur les automates temporisés suivants :

Automate M_i des état sanitaire de la parcelle i : Dans un modèle dit "bioéconomique" en cours de développement, nous définissons un état sanitaire (niveau d'épidémie) à l'échelle de la parcelle. Les résultats donnés ici ont été obtenus en considérant un risque sanitaire important en cas de traitement ancien et de l'occurrence d'une pluie. Ce risque peut être intégré à l'automate ci-dessous, et M_i n'est alors pas nécessaire.

Automate R_i règle à la parcelle i : Il décrit, en fonction du dernier traitement, du temps de rémanence du produit, des pluies prévues, la possibilité ou la nécessité de traiter une nouvelle fois. Il décrit aussi le stade phénologique à partir duquel il est opportun de commencer les traitements. On donne ici un exemple pour une règle Mildiou.

Automate phénologie B_i : On analyse suivant différents historiques de données. A l'heure actuelle, cet automate est un "automate ligne", qui produit une exécution unique avec les données du fichier choisi.

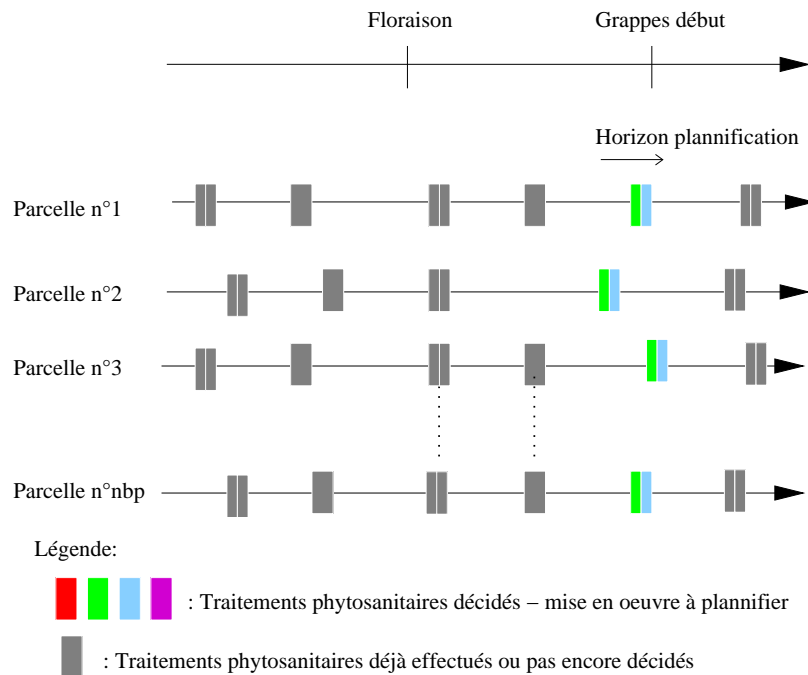


FIG. 4.3 – Horizon de planification

Automate météorologie (pluies prévues) P : Nous ne prenons pas en compte dans cet automate les températures, dont le cumul des effets se retrouve dans l'automate phénologie. Cet automate est un "automate ligne", qui produit une exécution unique avec les données du fichier choisi. L'alternative consisterait en un automate cyclique et stochastique qui réaliserait une "synthèse" des données historiques disponibles.

Obtention des fenêtres temporelles grâce à l'outil KRONOS

(Cette procédure est tirée du rapport de Tu Tuitete [Tui06])

Il s'agit d'obtenir pour une parcelle i donnée les dates de début et de fin de traitement de celle-ci. Pour cela nous devons poser une requête d'atteignabilité du sommet vérifiant le fait que la parcelle est en danger (prédicat TTN) et une valuation de l'horloge pour la rémanence (rem_i) supérieure à 10 (i.e. fin de rémanence) au produit de l'automate règle de la parcelle et de celui de prévision des pluies. Plus, concrètement, les lignes de commandes pour KRONOS sont les suivantes :

- Pour obtenir le produit des automates dans le fichier "*Resi.tg*" :
 $Kronos - outRes_i.tgpluiePrevue.tg regle_i.tg$
- Pour obtenir la date du prochain traitement de la parcelle i la formule TCTL est :
 $init \Rightarrow \exists \diamond (TTN \text{ and } Rem_i > 10)$

Après l'exécution de la deuxième commande, KRONOS nous fournit en sortie un fichier nommé *dfs.trace* contenant le parcours de tous les chemins avec les valuations des horloges correspon-

dantes dans l'automate "res" résultant du produit $pluiePrevue \times regle_i$. Ces parcours donnent les diverses transitions prises pour atteindre finalement le ou les sommets vérifiant $TTNandRem_i > 10$. Au final, nous sommes juste intéressés par la date échue. Les fichiers générés par Kronos doivent cependant au préalable être analysés et transformés en résultats exploitables. (voir pour cela le rapport [Tui06])

Le problème posé est ensuite de déterminer l'ordre dans lequel effectuer les traitements, et de grouper les parcelles à traiter dans des lots de travail. En effet, la cuve de pulvérisation a une capacité limitée et permet de traiter un nombre donné d'hectares suivant le type de traitement.

Une méthode d'ordonnancement autour du model-checking et des automates temporisés a été mis en œuvre par Y. Abdeddaim sur le problème de "job-shop" [Abd02]. Dans ses travaux Y. Abdeddaim représente un "job" par un automate temporisé comme représenté dans la figure Elle considère qu'une tâche m peut prendre trois états.

\bar{m} : La tâche est prête à être exécutée.

m : La tâche est en train d'être exécutée.

\underline{m} : La tâche a été exécutée.

Grâce à ces états, elle peut définir les labels des états de l'automate temporisé et mettre les contraintes temporelles, associées à la durée de la tâche, sur les transitions et dans les états de l'automate. Une fois ce travail effectué, elle fait un produit d'automate (défini dans le chapitre 2 section 2.2.1) où elle interdit la création des états qui auraient deux labels de type m et auquel elle ajoute une horloge global d . Ensuite elle vérifie à l'aide du model-checking la propriété d'atteignabilité de l'état *final* à partir de l'état initial. Elle énonce utiliser une propriété d'*immediat run* qui consiste à quitter un état dès que cela est possible. Cela lui permet de pratiquer un model-checking CTL et donc d'éviter la construction de l'automate des régions. De ce fait elle peut utiliser le model-checker UPPAAL étant donné qu'elle cherche à vérifier une propriété d'atteignabilité. Utiliser cette propriété d'*immediat run* lui évite également d'avoir dans les séquences explorées par le model-checker, des séquences où il n'y aurait aucunes tâches actives entre deux exécutions. Cela lui permet d'avoir le temps exact nécessaire à l'exécution des tâches suivant l'ordre dans lequel elles sont exécutées. Pour avoir l'ordonnancement optimal, il ne lui reste plus qu'à prendre la séquence dans laquelle la valeur de l'horloge d est la plus petite, et d'exécuter les tâches dans l'ordre où elles apparaissent dans cette séquence. Ce type d'ordonnancement a des limites que nous expliquerons dans le chapitre 6.

Nous avons donc choisi d'appliquer des méthodes d'ordonnancement plus conventionnelles que nous présentons dans le chapitre 5.

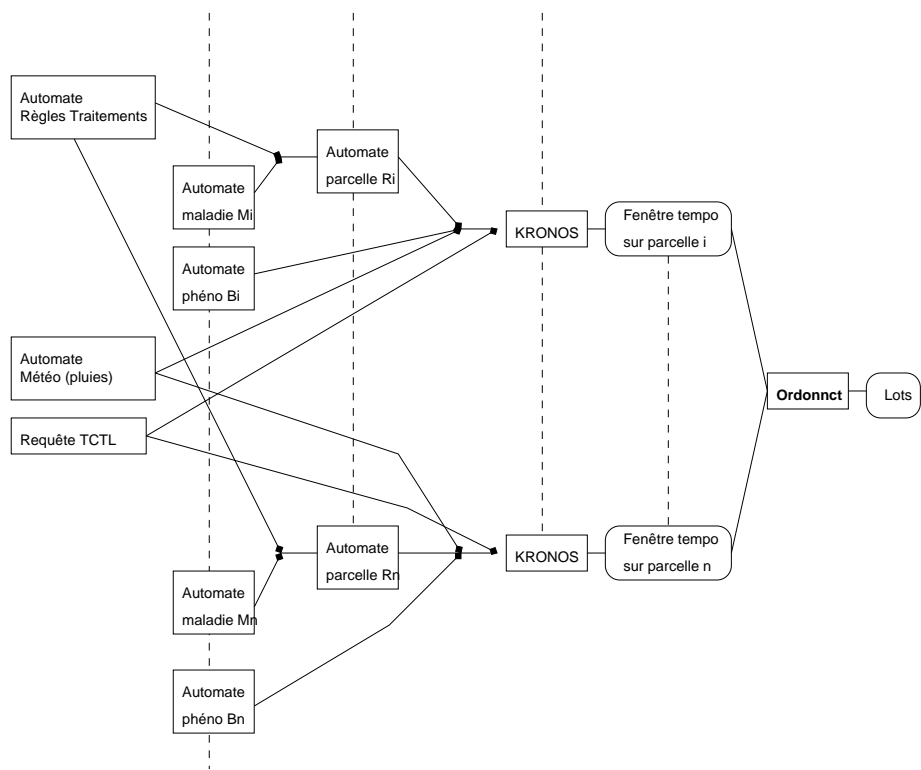


FIG. 4.4 – Méthode de résolution détaillée

Chapitre 5

Ordonnancement

Nous présenterons dans ce chapitre le problème d'ordonnancement (section 5.1), l'heuristique employée pour la confection de lots (5.2), un programme linéaire (5.3) et trois modélisations de programmation par contrainte (5.4).

5.1 Problème et notations

Nous devons déterminer l'ordre dans lequel effectuer les traitements, et grouper les parcelles à traiter dans des lots de travail, dans le but de réduire le temps total de travail. Le temps total de travail varie en fonction de l'ordre dans lequel les parcelles sont traitées (temps de transport entre parcelles). Donc, vérifier la faisabilité des traitements planifiés, suppose implicitement de proposer un ordonnancement couplé à un problème de voyageur de commerce.

Nous proposons plusieurs méthodologies pour résoudre ce problème. La première est basée sur la programmation linéaire en nombres entiers et est exposée en section 5.3. La seconde est basée sur la programmation par contraintes en utilisant le package CHOCO.

Nous introduisons nos notations dans les figures 5.1 et 5.2.

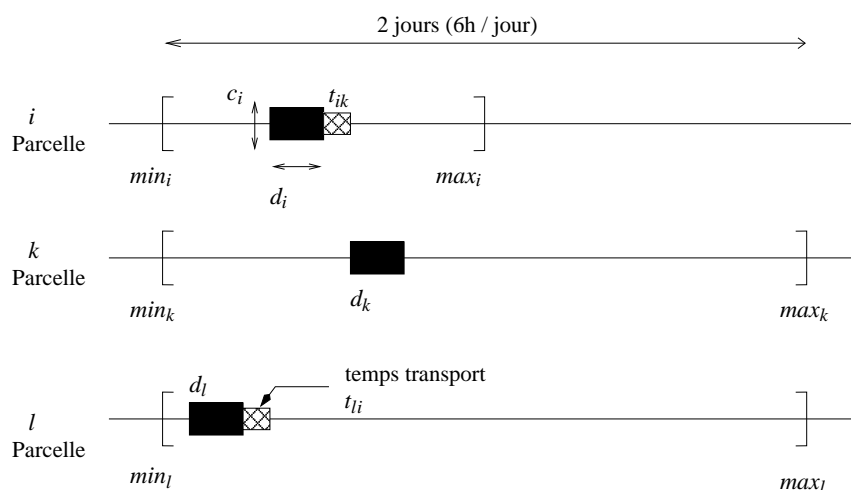


FIG. 5.1 – Représentation temporelle du traitement des parcelles

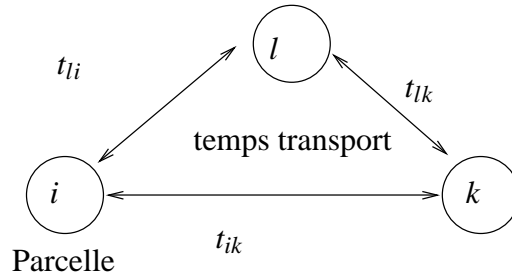


FIG. 5.2 – Voyageur de commerce

5.2 Heuristique employée pour la confection de lots et notations

Vu l'explosion combinatoire de notre problème d'ordonnancement, il nous est paru nécessaire d'organiser les parcelles par lots. Cela correspond à notre problème concret, où une série de parcelles est traitée après avoir rempli la cuve de produit. Pour cela nous mettons en place une heuristique générale de confection de lot que nous exposons après avoir défini les notations.

Notations

Nous utiliserons les notations suivantes :

nbp : nombre de parcelles,

nbl : nombre de lots,

Ct : quantité de produit contenue de la cuve du tracteur,

c_i : quantité de produit nécessaire pour traiter la parcelle i ,

d_i : durée du traitement de la parcelle i ,

D_j : durée de traitement du lot j ,

max_i : borne supérieure de la fenêtre temporelle de la parcelle i ,

min_i : borne inférieure de la fenêtre temporelle de la parcelle i ,

Max_j : borne supérieure de la fenêtre temporelle du lot j ,

Min_j : borne inférieure de la fenêtre temporelle du lot j ,

t_{ik} : durée de trajet entre la parcelle i et la parcelle k .

L'heuristique

- 1 La fenêtre temporelle d'un lot est l'intersection des fenêtres temporelles des parcelles qui composent ce lot.
- 2 La somme des durée des tâches effectuées sur les parcelles appartenant à un même lot doit être inférieure à la différence entre la borne supérieure et la borne inférieure de la fenêtre temporelle de ce lot.
- 3 La somme des quantité de produit utilisé des parcelles appartenant à un lot doit être inférieure à la quantité de produit contenue dans une cuve de tracteur et supérieure à $\frac{2Ct}{3}$.

5.3 Programmation linéaire en nombres entiers

Dans un premier temps, nous allons décrire un programme linéaire en nombres entiers permettant de donner la solution optimale à notre problème de confection de lots avec l'heuristique présentée en section 5.2. Nous proposerons ensuite une idée d'algorithme d'approximation.

5.3.1 Programme linéaire en nombres entiers

Nous proposons ici une formulation du problème de confection de lots comme un programme linéaire en nombres entiers.

Pour modéliser le fait qu'une parcelle appartienne à un lot, nous utilisons le formalisme suivant :

$$X_{ij} = \begin{cases} 1, & \text{si la parcelle } i \text{ appartient au lot } j \\ 0, & \text{sinon.} \end{cases}$$

Pour modéliser le fait que deux parcelles d'un même lot seront traitées l'une à la suite de l'autre, nous utilisons le formalisme suivant :

$$Y_{ij} = \begin{cases} 1, & \text{si la parcelle } i \text{ est prédecesseur de la parcelle } j \\ 0, & \text{sinon.} \end{cases}$$

Entre chaque lots le tracteur doit repasser à la ferme pour remplir sa cuve. Nous ajoutons donc le formalisme suivant :

$$Y_{iF} = \begin{cases} 1, & \text{si la parcelle } i \text{ est prédecesseur de la ferme} \\ 0, & \text{sinon.} \end{cases}$$

$$Y_{Fi} = \begin{cases} 1, & \text{si la ferme est prédecesseur de la parcelle } i \\ 0, & \text{sinon.} \end{cases}$$

Nous souhaitons respecter certaines contraintes, avant tout un lot ne devra pas avoir besoin de plus d'une cuve de produit pour être traité, nous pouvons donc introduire les contraintes suivantes :

$$\sum_{i=1}^{nbp} C_i \times X_{ij} \leq Ct \quad \forall j \in [0, nbl]$$

De plus, pour des raisons économiques nous souhaitons ne pas créer de lot qui aurait besoin de moins de deux tiers d'une cuve pour être traité, nous avons les contraintes suivantes :

$$\sum_{i=1}^{nbp} C_i \times X_{ij} \geq \frac{2Ct}{3} \quad \forall j \in [0, nbl]$$

D'après notre heuristique, nous avons choisi de prendre comme fenêtre temporelle d'un lot ***l'intersection*** des fenêtres temporelles des parcelles qui le composent, nous posons donc les égalités suivantes :

$$\max(\min_i \times X_{ij}) = Min_j \quad \forall j \in [0, nbl]$$

$$\min(\max_i \times (\frac{1}{X_{ij} + \epsilon})) \times \frac{1}{X_{ij} + \epsilon} = Max_j^1 \quad \forall j \in [0, nbl], \epsilon > 0$$

¹Si nous posons $\min(\max_i X_{ij}) = Max_j$ nous obtiendrons 0 comme valeur pour Max_j dans le cas où une parcelle n'appartiendrait pas au lot j . Or la borne supérieure de la fenêtre temporelle du lot j est, d'après notre heuristique, la borne supérieure de l'intersection des fenêtres temporelles des parcelles qui le composent. Avec cette manipulation mathématique nous évitons ce problème en augmentant la borne supérieure de la fenêtre temporelle d'une parcelle n'appartenant pas au lot j lors de la sélection du minimum. ϵ étant un réel positif très proche de 0.

Nous décidons également d'imposer que les traitements des parcelles composant le même lot soient effectués dans la fenêtre temporelle de ce lot, ce qui nous donne les contraintes suivantes :

$$\sum_{i=1}^{nbp} d_i \times X_{ij} \leq (Max_j - Min_j) \quad \forall j \in [0, nbl]$$

Or nous devons prendre en compte la durée de parcours entre deux parcelles du même lot, (nous considérons ici que si la durée du parcours de la parcelle i à la parcelle k est égale à d alors la durée du parcours de la parcelle k à la parcelle i est elle aussi égale à d) nous arrivons donc aux contraintes suivantes :

$$\sum_{i=1}^{nbp} \left(d_i \times X_{ij} + \sum_{k=1}^{nbp} (t_{ik} \times X_{ij} \times X_{kj} \times \mathcal{Y}_{ik}) + t_{iF} \times \mathcal{Y}_{iF} + t_{Fi} \times \mathcal{Y}_{Fi} \right) \leq (Max_j - Min_j) \quad \forall j \in [0, nbl]$$

Nous souhaitons qu'une parcelle k soit précédée d'une seule autre parcelle i ou de la ferme. Nous modélisons ce fait par les contraintes suivantes :

$$\mathcal{Y}_{Fk} + \sum_{i=1}^{nbp} \mathcal{Y}_{ik} \geq 1 \quad \forall k \neq i; k \in [0, nbp]$$

Nous souhaitons qu'une parcelle k soit suivie d'une seule autre parcelle i ou de la ferme. Nous modélisons ce fait par les contraintes suivantes :

$$\mathcal{Y}_{kF} + \sum_{i=1}^{nbp} \mathcal{Y}_{ki} \geq 1 \quad \forall k \neq i; k \in [0, nbp]$$

Pour garder une cohérence dans la succession des parcelles nous ajoutons les contraintes suivantes :

$$\mathcal{Y}_{kF} + \mathcal{Y}_{Fk} = 1 \quad \forall k \in [0, nbp]$$

$$\mathcal{Y}_{ki} + \mathcal{Y}_{ik} = 1 \quad \forall k \neq i; i, k \in [0, nbp]$$

Une parcelle doit obligatoirement appartenir à un lot, et avoir une parcelle comme successeur ou une parcelle comme prédécesseur dans ce lot :

$$\sum_{j=0}^{nbl} \sum_{k=0}^{nbp} (X_{ij} \times (\mathcal{Y}_{ik} + \mathcal{Y}_{ki}) \times X_{kj}) \geq 1 \quad \forall i \in [0, nbp]$$

Nous choisissons de minimiser la durée de travail totale (traitement + trajet), donc de minimiser cette équation :

$$\min(z) = \sum_{j=1}^{nbl} \sum_{i=1}^{nbp} \left(d_i \times X_{ij} + \sum_{k=1}^{nbp} (t_{ik} \times X_{ij} \times X_{kj} \times \mathcal{Y}_{ik}) + t_{iF} \times \mathcal{Y}_{iF} + t_{Fi} \times \mathcal{Y}_{Fi} \right)$$

On peut désormais formuler notre problème de confection de lots, en un problème de program-

mation linéaire en nombres entiers :

$$\mathcal{PLNE} \left\{ \begin{array}{l}
 \min(z) = \sum_{j=1}^{nbl} \sum_{i=1}^{nbp} \left(d_i \times X_{ij} + \sum_{k=1}^{nbp} (t_{ik} \times X_{ij} \times X_{kj} \times \mathcal{Y}_{ik}) + t_{iF} \times \mathcal{Y}_{iF} + t_{Fi} \times \mathcal{Y}_{Fi} \right) \\
 \mathcal{Y}_{Fk} + \sum_{i=1}^{nbp} \mathcal{Y}_{ik} \geq 1 \\
 \mathcal{Y}_{kF} + \sum_{i=1}^{nbp} \mathcal{Y}_{ki} \geq 1 \\
 \sum_{j=0}^{nbl} \sum_{k=0}^{nbp} (X_{ij} \times (\mathcal{Y}_{ik} + \mathcal{Y}_{ki}) \times X_{kj}) \geq 1 \\
 \sum_{i=1}^{nbp} C_i \times X_{ij} \leq Ct \\
 \sum_{i=1}^{nbp} C_i \times X_{ij} \geq \frac{2Ct}{3} \\
 \min(\max_i \times (\frac{1}{X_{ij} + \epsilon})) \times \frac{1}{X_{ij} + \epsilon} = Max_j \\
 \max(\min_i \times X_{ij}) = Min_j \\
 \sum_{i=1}^{nbp} \left(d_i \times X_{ij} + \sum_{k=1}^{nbp} (t_{ik} \times X_{ij} \times X_{kj} \times \mathcal{Y}_{ik}) + t_{iF} \times \mathcal{Y}_{iF} + t_{Fi} \times \mathcal{Y}_{Fi} \right) \leq (Max_j - Min_j) \\
 \mathcal{Y}_{kF} + \mathcal{Y}_{Fk} = 1 \\
 \mathcal{Y}_{ki} + \mathcal{Y}_{ik} = 1 \\
 X_{ij}, \mathcal{Y}_{ik}, \mathcal{Y}_{iF}, \mathcal{Y}_{Fi} \in \{0, 1\}
 \end{array} \right.$$

5.3.2 Algorithme d'approximation

Dans un premier temps, sachant la résolution d'un PLNE (cad la recherche d'une solution satisfaisant les contraintes et minimisant (ici) une fonction objectif, est un probleme NP-complet, nous allons procéder à une relaxation des contraintes d'intégrité en prenant les variables bivalentes non plus dans l'ensemble 0,1 mais dans l'intervalle 0,1, nous obtenons donc un programme linéaire (PL). Ensuite, comme nous voulons obtenir une solution à valeurs entières, nous allons utiliser la procédure suivante : Nous résoudrons ce programme linéaire (PL) puis nous y appliquerons un algorithme d'approximation qui travaillerait sur sa solution optimale. Le principe serait de trier les X_{ij} par ordre décroissant puis de les prendre un par un et de mettre la parcelle i dans le lot j en vérifiant juste de ne pas dépasser la capacité de la cuve du tracteur. Si la capacité de la cuve est dépassée alors nous recherchons le prochain X_{lk} dans la liste tel que $l = i$ et $k \neq j$ auquel nous faisons subir le même traitement. Si aucun X_{lk} n'a permis de mettre la parcelle i dans un lot alors nous créerons un lot j' pour y mettre la parcelle i . Une fois la parcelle i attribuée dans un lot nous supprimons de la liste tous les X_{lk} tel que $l = i$. Nous appliquons ce traitement à chaque X_{ij} jusqu'à ce que la liste soit vide. Une fois les parcelles attribuées dans les lots nous pourrions effectuer un traitement des \mathcal{Y}_{ik} pour affecter à une parcelle son successeur. Par manque de temps, nous n'avons pas pu traiter cette partie lors de ce stage.

5.4 Programmation par contraintes

Lors de la confection de lots, nous avons plusieurs contraintes à respecter pour que l'ordonnement soit toujours réalisable. Cette présence de contraintes nous a poussé tout naturellement vers la programmation par contrainte. Nous allons détailler dans cette section les concepts de modélisation et la modélisation dans CHOCO².

Nous ajoutons les notations suivantes :

C_i : quantité de produit nécessaire pour traiter le lot i ,

D_i : durée de travail sur le lot i ,

²<http://choco-solver.net>

$$d_{ij} \begin{cases} d_i, & \text{si la parcelle } i \text{ appartient au lot } j \\ 0, & \text{sinon.} \end{cases}$$

$$c_{ij} \begin{cases} c_i, & \text{si la parcelle } i \text{ appartient au lot } j \\ 0, & \text{sinon.} \end{cases}$$

$$min_{ij} \begin{cases} min_i, & \text{si la parcelle } i \text{ appartient au lot } j \\ 0, & \text{sinon.} \end{cases}$$

$$max_{ij} \begin{cases} max_i, & \text{si la parcelle } i \text{ appartient au lot } j \\ 0, & \text{sinon.} \end{cases}$$

Nous complétons l'heuristique (définie page 30) pour la confection de lots :

- 1 La fenêtre temporelle d'un lot est toujours l'intersection des fenêtres temporelles des parcelles qui le composent.
- 2 On ajoute à chaque durée de traitement d_i la plus grande durée de trajet t_{ik} entre la parcelle i et les autres parcelles, nous appelons cette durée avec trajet : dt_i .
- 3 La somme des dt_i des parcelles appartenant à un même lot doit être inférieure à la différence entre la borne supérieure et la borne inférieure de la fenêtre temporelle de ce lot.

5.4.1 Concepts de modélisation

Nous allons présenter ici deux types de modélisation, que nous appellerons "Type A" et "Type B". Pour la modélisation "Type A", nous avons mis au point deux programmes (V1 et V1.1) et un programme (V2) pour la modélisation "Type B".

Modélisation "Type A"

Dans la modélisation de "Type A", nous calculons le nombre de lots voulu de la façon suivante :

$$nbl = \left\lceil \frac{\sum_{i=0}^{nbp} c_i}{Ct} \right\rceil + 1$$

Nous allons créer nbl lots qui auront chacun une fenêtre temporelle ($[Min_i, Max_i]$), une durée de travail D_i , une quantité de produit utilisé C_i . La fenêtre temporelle sera l'intersection des fenêtres temporelles des parcelles composant ce lot. La durée de travail sera la somme des durées de traitement des parcelles composant ce lot, plus la somme des durées de transport entre ces parcelles. La quantité de produit utilisé sera la somme des quantités de produit utilisé des parcelles composant ce lot. Avec cette modélisation, les fenêtres temporelles de nos lots sont créées dynamiquement, ce qui rend cette modélisation plus générale et applicable dans un domaine industriel plus large. Nous verrons que certaines instances de notre problème n'ont pas de solution avec cette modélisation du fait du nombre de lots voulus trop bas.

Le concept de modélisation de "Type A" est représenté dans la figure 5.3.

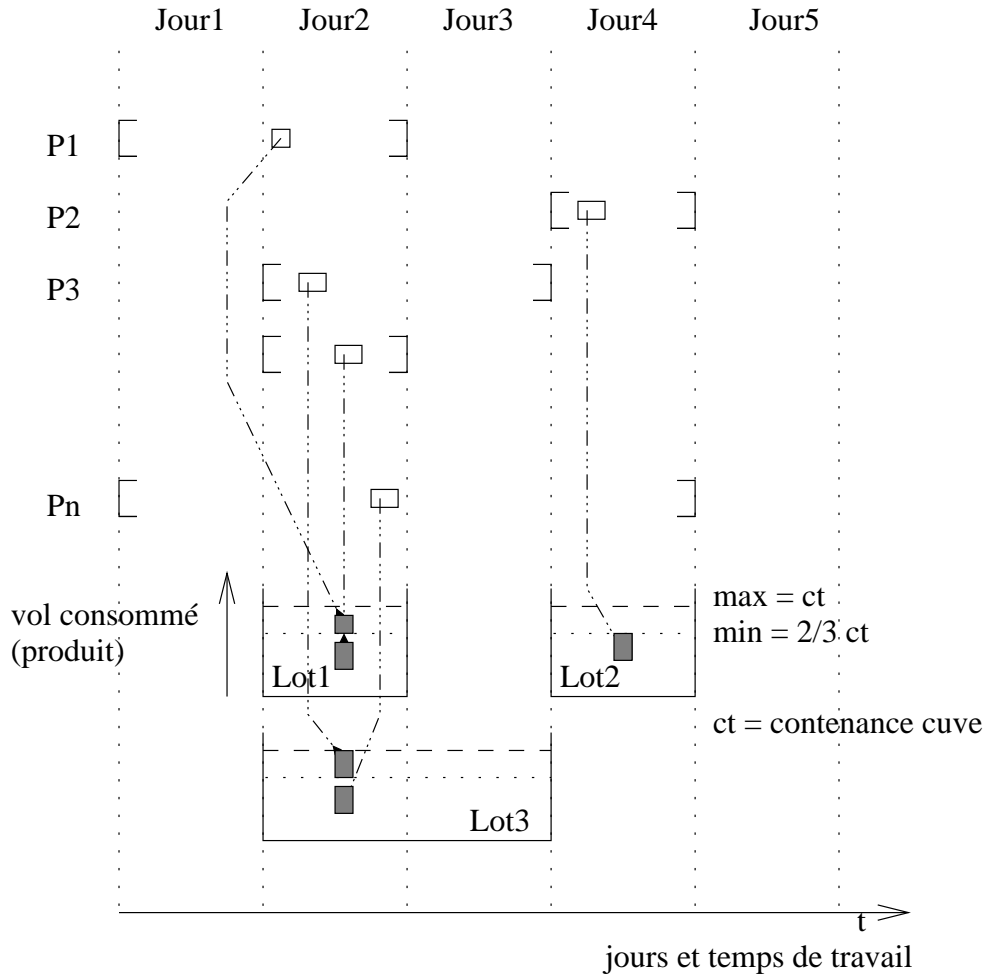


FIG. 5.3 – Représentation du concept de modélisation de "Type A"

Modélisation "Type B"

La modélisation de "Type B" diffère de la modélisation de "Type A" du fait que les fenêtres temporelles des lots ne sont pas calculées. Cette modélisation se rapproche plus du problème réel. Le nombre de lot est défini par l'horizon de prévoyance donné en jour, un jour correspondra à un lot. Nous pouvons poser cette contrainte sans pour autant perdre de complétude dans la modélisation de notre problème car une parcelle n'aura pas une durée de traitement supérieure à une journée. Nous considérons également que si une parcelle peut être traitée le jour i elle peut être traitée à n'importe quel moment de la journée, en effet étant dans un environnement incertain (météorologie) nous ne pouvons pas prévoir exactement à quel moment de la journée il va pleuvoir sur une parcelle et donc nous considérons que cette parcelle ne pourra pas être traitée ce jour. En voyant notre problème sous cet angle, nous pouvons aisément redéfinir les fenêtres temporelles de nos parcelles par une liste de jours durant lesquels elles pourront être traitées (ce qui représente la liste des lots auxquels elle peut appartenir). Comme dans la modélisation de "Type A", la durée de travail D_i d'un lot sera la somme des durées de traitement des parcelles composant ce lot, plus la somme des durées de transport entre ces parcelles. La quantité de produit utilisé C_i pour un lot sera la somme des quantités de produit utilisé des parcelles composant ce lot. Nous mettons en place le système de pénalité suivant :

- Si un lot est créé nous infligeons une pénalité de 10.
- Si un lot est créé avec une quantité de produit inférieure à $\frac{2Ct}{3}$ nous infligeons une pénalité

supplémentaire de 10.

Le concept de modélisation de "Type B" est représenté dans la figure 5.4.

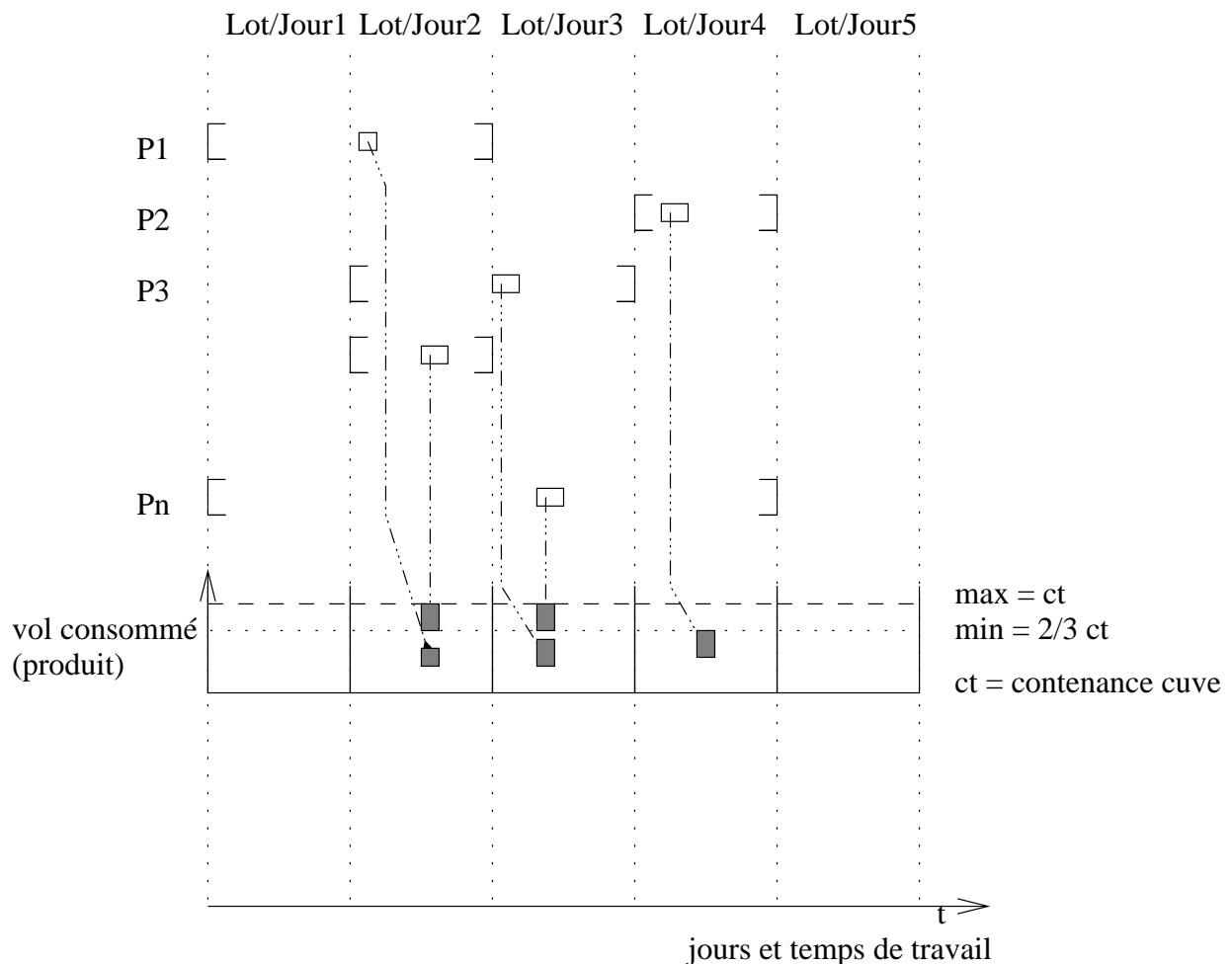


FIG. 5.4 – Représentation du concept de modélisation de "Type B"

Maintenant il ne nous reste plus qu'à minimiser pénalité pour avoir les lots nécessaire, ainsi que la répartition de nos parcelles dans ces lots, pour traité notre exploitation en respectant au mieux ses contraintes.

5.4.2 La modélisation dans CHOCO

CHOCO³ est une bibliothèque de programmation par contraintes (PPC). CHOCO rassemble, dans une architecture orientée-objets, les principaux utilitaires des systèmes de PPC, comme les structures de données implémentant des domaines de valeurs, les événements de propagation, des algorithmes de filtrage ou encore la recherche arborescente. L'implémentation a été faite avec deux objectifs : d'une part offrir une architecture logicielle modulaire qui accepte aisément des extensions, et d'autre part, mettre en œuvre une gestion optimisée des événements de propagation.

³<http://choco-solver.net>

Programme V1 ("Type A")

Les variables

Nous détaillons dans cette section les variables utilisées pour modéliser notre problème en un CSP⁴.

Les durées : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les durées de traitement de lots.

$$\boxed{d_{11} \quad \cdots \quad \cdots \quad \cdots \quad d_{n1} \quad || \quad D_1} \quad \cdots \quad \boxed{d_{1k} \quad \cdots \quad \cdots \quad \cdots \quad d_{nk} \quad || \quad D_k}$$

avec pour domaine

- $D_j : [0, \sum_{i=0}^{nbp} d_i]$.
- $d_{ij} : [0, d_i]$.

Les quantités de produit : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les quantités de produit utilisées par lots.

$$\boxed{c_{11} \quad \cdots \quad \cdots \quad \cdots \quad c_{n1} \quad || \quad C_1} \quad \cdots \quad \boxed{c_{1k} \quad \cdots \quad \cdots \quad \cdots \quad c_{nk} \quad || \quad C_k}$$

Nous savons qu'un lot ne doit pas avoir besoin de plus de produit que la cuve ne peut en contenir, de plus pour des raisons économiques nous ne souhaitons pas créer de lot ayant besoin de moins de deux tiers de cuve pour être traité. Nous intégrons donc ces contraintes directement dans le domaine de nos variables.

- $C_j : [\frac{2C_t}{3}, C_t]$.
- $c_{ij} : [0, c_i]$.

Les bornes inférieures : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les bornes inférieures des fenêtres temporelles des lots.

$$\boxed{min_{11} \quad \cdots \quad \cdots \quad \cdots \quad min_{n1} \quad || \quad Min_1} \quad \cdots \quad \boxed{min_{1k} \quad \cdots \quad \cdots \quad \cdots \quad min_{nk} \quad || \quad Min_k}$$

avec pour domaine

- $min_j : [0, \max(min_i)]$.
- $min_{ij} : [0, min_i]$.

Les bornes supérieures : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les bornes supérieures des fenêtres temporelles des lots.

$$\boxed{max_{11} \quad \cdots \quad \cdots \quad \cdots \quad max_{n1} \quad || \quad Max_1} \quad \cdots \quad \boxed{max_{1k} \quad \cdots \quad \cdots \quad \cdots \quad max_{nk} \quad || \quad Max_k}$$

avec pour domaine

- $max_j : [0, \max(max_i)]$.
- $max_{ij} : [0, max_i]$.

⁴Constraint Satisfaction Problems

Les parcelles : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter l'appartenance ou pas de la parcelles i au lot j . La variable p_{ij} aura pour valeur 1 si la parcelle i appartient au lot j , 0 sinon.

$$\boxed{p_{11} \quad \cdots \quad \cdots \quad \cdots \quad p_{nbp \setminus nbl}}$$

avec pour domaine

$$- p_{ij} : [0, 1].$$

Les contraintes

Nous allons détailler les contraintes portant uniquement sur un type de variable (parcelles, quantité utilisé, durée, borne supérieure, borne inférieure) puis les contraintes entre ces types.

Contrainte sur les parcelles : En premier lieu nous devons interdire à une parcelle d'appartenir à deux lots. Nous posons donc les contraintes suivante :

$$\sum_{i=0}^{nbp} \sum_{j=0}^{nbl} p_{ij} = nbp$$

$$\text{si } p_{ij} = 1 \text{ alors } p_{ik} = 0 \quad \forall k \neq j; j, k \in [0, nbl]$$

Contrainte sur les durées : La durées de traitement d'un lot est égal à la somme des d_i des parcelles appartenant à ce lot.

$$D_j = \sum_{i=0}^k d_{ij} \quad \forall j \in [0, nbl]$$

Contrainte sur les quantités de produit : La quantité de produit utilisée pour traiter un lot est égal à la somme des c_i des parcelles appartenant à ce lot.

$$C_j = \sum_{i=0}^k c_{ij} \quad \forall j \in [0, nbl]$$

Contrainte sur les bornes inférieures : La borne inférieure d'un lot est égal à la plus grande borne inférieure des parcelles appartenant à ce lot.

$$Min_j = \max(\min_{ij}) \quad \forall j \in [0, nbl]$$

Contrainte sur les bornes supérieures : La borne supérieure d'un lot est égal à la plus petite borne supérieure des parcelles appartenant à ce lot.

$$Max_j = \min(\max_{ij}) \quad \forall j \in [0, nbl]$$

Contraintes entre les parcelles et les autres variables : Pour mettre à jour nos variables représentant les durées (d_{ij}), les bornes supérieures (\max_{ij}), les bornes inférieures (\min_{ij}) et les quantités de produit (c_{ij}) nous posons les contraintes suivantes :

$$d_{ij} = \begin{cases} d_i, & \text{si } p_{ij} = 1 \\ 0, & \text{sinon.} \end{cases}$$

$$c_{ij} = \begin{cases} c_i, & \text{si } p_{ij} = 1 \\ 0, & \text{sinon.} \end{cases}$$

$$min_{ij} = \begin{cases} min_i, & \text{si } p_{ij} = 1 \\ 0, & \text{sinon.} \end{cases}$$

Pour les bornes supérieures étant donné que d'après notre heuristique nous choisissons de prendre comme fenêtre temporelle d'un lot *l'intersection* des fenêtres temporelles des parcelles qui le composent, nous prenons $\max(max_i)$ comme valeur pour un max_{ij} si $p_{ij} = 0$ et non pas 0. Si nous ne faisons pas cela, lorsque nous chercherons la borne supérieure de la fenêtre temporelle d'un lot, nous trouverons forcément 0 comme valeur. Pour éviter cela, nous posons les contraintes suivantes :

$$max_{ij} = \begin{cases} max_i, & \text{si } p_{ij} = 1 \\ \max(max_i), & \text{sinon.} \end{cases}$$

Contraintes entre les fenêtres temporelles et les durées : Nous imposons que la durée de traitement d'un lot soit inférieure à la norme de sa fenêtre temporelle.

$$D_j \leq (Max_j - Min_j) \quad \forall j \in [0, nbl]$$

Nous obtenons pour une exploitation contenant 14 parcelles et 3 lots :

Nombres de variables : 223

Nombres de contraintes : 472

Programme V1.1 ("Type A")

Les variables

Nous détaillons dans cette section les variables utilisées pour modéliser notre problème en un CSP⁵.

Les quantités de produit : Nous utilisons un tableau de IntVar de taille nbl pour représenter les quantités de produit utilisées par lots.

C_1	C_{nbl}
-------	-----	-----	-----	-----------

Nous savons qu'un lot ne doit pas avoir besoin de plus de produit que la cuve ne peut en contenir, de plus pour des raisons économiques nous ne souhaitons pas créer de lot ayant besoin de moins de deux tiers de cuve pour être traité. Nous intégrons donc ces contraintes directement dans le domaine de nos variables.

$$- C_j : \left[\frac{2C_j}{3}, C_j \right].$$

Les bornes inférieures : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les bornes inférieures des fenêtres temporelles des lots.

min_{11}	min_{n1}		Min_1	min_{1k}	min_{nk}		Min_k
------------	-----	-----	-----	------------	--	---------	-----	-----	------------	-----	-----	-----	------------	--	---------

avec pour domaine

$$- min_j : [0, \max(min_i)].$$

⁵Constraint Satisfaction Problems

– $min_{ij} : [0, min_i]$.

Les bornes supérieures : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les bornes supérieures des fenêtres temporelles des lots.

max_{11}	max_{n1}		Max_1	max_{1k}	max_{nk}		Max_k
------------	-----	-----	-----	------------	--	---------	-----	-----	------------	-----	-----	-----	------------	--	---------

avec pour domaine

- $max_j : [0, max(max_i)]$.
- $max_{ij} : [0, max_i]$.

Les parcelles : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter l'appartenance ou pas de la parcelles i au lot j . La variable p_{ij} aura pour valeur 1 si la parcelle i appartient au lot j , 0 sinon.

p_{11}	$p_{nbp \setminus nbl}$
----------	-----	-----	-----	-------------------------

avec pour domaine

- $p_{ij} : [0, 1]$.

Les contraintes

Nous allons détailler les contraintes portant uniquement sur un type de variable (parcelles, borne supérieure, borne inférieure) puis les contraintes entre ces types.

Contrainte sur les parcelles : En premier lieu nous devons interdire à une parcelle d'appartenir à deux lots. Nous posons donc les contraintes suivante :

$$\sum_{i=0}^{nbp} \sum_{j=0}^{nbl} p_{ij} = nbp$$

$$\text{si } p_{ij} = 1 \text{ alors } p_{ik} = 0 \quad \forall k \neq j; j, k \in [0, nbl]$$

Contrainte sur les bornes inférieures : La borne inférieure d'un lot est égal à la plus grande borne inférieure des parcelles appartenant à ce lot.

$$Min_j = max(min_{ij}) \quad \forall j \in [0, nbl]$$

Contrainte sur les bornes supérieures : La borne supérieure d'un lot est égal à la plus petite borne supérieure des parcelles appartenant à ce lot.

$$Max_j = min(max_{ij}) \quad \forall j \in [0, nbl]$$

Contraintes entre les parcelles et les autres variables : Pour mettre à jour nos variables représentant les bornes supérieures (max_{ij}), les bornes inférieures (min_{ij}) et les quantités de produit (C_j) nous posons les contraintes suivantes :

$$min_{ij} = \begin{cases} min_i, & \text{si } p_{ij} = 1 \\ 0, & \text{sinon.} \end{cases}$$

Pour les bornes supérieures étant donné que d'après notre heuristique nous choisissons de prendre comme fenêtre temporelle d'un lot l'*intersection* des fenêtres temporelles des parcelles qui le composent, nous prenons $\max(\max_i)$ comme valeur pour un \max_{ij} si $p_{ij} = 0$ et non pas 0. Si nous ne faisons pas cela, lorsque nous chercherons la borne supérieure de la fenêtre temporelle d'un lot, nous trouverons forcément 0 comme valeur. Pour éviter cela, nous posons les contraintes suivantes :

$$\max_{ij} = \begin{cases} \max_i, & \text{si } p_{ij} = 1 \\ \max(\max_i), & \text{sinon.} \end{cases}$$

$$C_j = \sum_{i=0}^{nbp} c_i \times p_{ij} \quad \forall j \in [0, nbl]$$

Contraintes entre les fenêtres temporelles et les durées : Nous imposons que la durée de traitement d'un lot soit inférieure à la norme de sa fenêtre temporelle.

$$\sum_{i=0}^{nbp} d_i \times p_{ij} \leq (Max_j - Min_j) \quad \forall j \in [0, nbl]$$

Nous obtenons pour une exploitation contenant 14 parcelles et 3 lots :

Nombres de variables : 135

Nombres de contraintes : 343

Programme V2 ("Type B")

Les variables

Nous détaillons dans cette section les variables utilisées pour modéliser notre problème en un CSP⁶.

Les quantités de produit : Nous utilisons un tableau de IntVar de taille $nbp + 1 \times nbl$ pour représenter les quantités de produit utilisées par lots.

$$\boxed{c_{11}} \quad \cdots \quad \cdots \quad \cdots \quad \boxed{c_{n1}} \quad \parallel \quad \boxed{C_1} \quad \cdots \quad \boxed{c_{1k}} \quad \cdots \quad \cdots \quad \cdots \quad \boxed{c_{nk}} \quad \parallel \quad \boxed{C_k}$$

Nous savons qu'un lot ne doit pas avoir besoin de plus de produit que la cuve ne peut en contenir, de plus pour des raisons économiques nous ne souhaitons pas créer de lot ayant besoin de moins de deux tiers de cuve pour être traité. Nous intégrons donc ces contraintes directement dans le domaine de nos variables.

- $C_j : [\frac{2Ct}{3}, Ct]$.
- $c_{ij} : [0, c_i]$.

Les parcelles : Nous utilisons un tableau de IntVar de taille $nbp + 1$ pour représenter l'appartenance ou pas de la parcelles i au lot j . La variable p_i aura pour valeur le numéro du lot auquel elle appartient.

$$\boxed{p_1} \quad \cdots \quad \cdots \quad \cdots \quad \boxed{p_{nbp}}$$

avec pour domaine

⁶Constraint Satisfaction Problems

- p_i aura pour domaine les jours durant lesquels elle pourra être traitée.

Variable de minimisation : Nous utilisons une variable supplémentaire dont nous chercherons à minimiser sa valeur avec pour domaine :

- $V : [0, +\infty]$.

Les contraintes

Contrainte sur les quantités de produit : La quantité de produit utilisée pour traiter un lot est égale à la somme des c_i des parcelles appartenant à ce lot plus une pénalité, 10 pour un lot contenant au moins une parcelle, 20 pour un lot contenant au moins une parcelle et utilisant une quantité de produit inférieure à $\frac{2Ct}{3}$.

$$C_j = \begin{cases} \sum_{i=0}^{nbp} c_{ij}, & \text{si } \sum_{i=0}^k c_{ij} = 0 \quad \forall j \in [0, nbl] \\ 10 + \sum_{i=0}^{nbp} c_{ij}, & \text{si } \sum_{i=0}^k c_{ij} \neq 0 \quad \forall j \in [0, nbl] \\ 20 + \sum_{i=0}^{nbp} c_{ij}, & \text{si } \sum_{i=0}^k c_{ij} \leq \frac{2Ct}{3} \quad \forall j \in [0, nbl] \end{cases}$$

Contraintes entre les parcelles et les quantités de produit : Pour mettre à jour nos variables représentant les quantités de produit (C_{ij}) nous posons les contraintes suivantes :

$$c_{ij} = \begin{cases} c_i, & \text{si } p_{ij} = 1 \\ 0, & \text{sinon.} \end{cases}$$

Contraintes sur la variable à minimiser : Nous souhaitons minimiser la somme des pénalités. Etant donné que la somme des quantités de produit utilisées est constante on pose la contrainte suivante :

$$V = \sum_{j=0}^{nbl} C_j \quad \forall j \in [0, nbl]$$

Nous obtenons pour une exploitation contenant 14 parcelles et 3 lots :

Nombres de variables : 90

Nombres de contraintes : 156

5.4.3 Résultats

Pour mettre en œuvre nos expériences nous avons créé deux heuristiques de tri sur les parcelles (H1 et H2) que nous détaillons plus loin, nous avons également créé leur inverse (\neg H1 et \neg H2). Nous travaillons sur une exploitation *compacte* (les parcelles ne sont pas éloignées les unes des autres). Nous notons par SansH le fait de n'avoir effectué aucun tri sur les parcelles, par T le temps en milliseconde de résolution et par N le nombre de nœuds parcouru dans l'arbre de recherche.

Heuristique de tri

H1 Effectue un tri sur la liste des parcelles en les classant par ordre croissant suivant le rapport $\frac{\max_i - \min_i}{c_i}$.

H2 Effectue un tri sur la liste des parcelles en les classant par ordre décroissant suivant $nbInter_i$, avec $nbInter_i$ le nombre d'intersection non nulle entre la fenêtre temporelle de la parcelle i et les fenêtres temporelles des autres parcelles.

Nous avons implémenté ces heuristiques dans le but de voir si il y avait un impact significatif sur le temps de résolution, de l'ordre dans lequel sont instanciées les parcelles. Ces heuristiques n'ont été implémentées que pour les programmes V1.1 et V2. Nous allons dans un premier temps énoncer la mise en œuvre de nos expérimentations, ensuite nous comparerons le programme V1 et le programme V1.1, qui sont basés sur la modélisation de "Type A" (vu en section 5.4.1), puis nous comparerons le programme V1.1 et le programme V2. Finalement nous verrons si il y a ou pas un impact significatif de nos heuristiques sur le temps de résolution.

Données utilisées pour les simulations

Nombre de lots ⁷ : $nbl = 3$

Nombre de parcelle : $nbp = 14$

Quantité de produit : contenu dans la cuve $C_t = 1200$

Horizon de provoyance : 5 jours

Somme des quantités de produits utilisées pour le traitement des parcelles : 2634

Une fois le programme V1 implémenté, nous avons lancé 75 simulations avec des valeurs aléatoires pour les fenêtres temporelles des parcelles. Parmi ces simulations nous en avons sélectionné 6, 3 simulations ayant une solution pour la modélisation de "Type A" et 3 autres n'ayant pas de solution.

S.3 La simulation numéro 3 est la simulation qui trouvait une solution et dont le temps de résolution était le plus petit

S.12 La simulation numéro 12 est une simulation qui trouvait une solution et dont le temps de résolution était dans la gaussienne des simulations ayant une solution.

S.36 La simulation numéro 36 est une simulation qui trouvait une solution et dont le temps de résolution était le plus grand.

S.17 La simulation numéro 17 est une simulation, n'ayant pas de solution, dont le temps de résolution était le plus grand.

S.59 La simulation numéro 59 est une simulation, n'ayant pas de solution, dont le temps de résolution était dans la gaussienne des simulations n'ayant pas de solution.

S.69 La simulation numéro 69 est la simulation, n'ayant pas de solution, dont le temps de résolution était le plus petit

Nous avons ensuite gardé les données (fenêtres temporelles) de ces simulations pour le reste de nos expérimentations.

V1 ("Type A") vs V1.1 ("Type A")

En comparant le tableau 5.1 et le tableau 5.2, nous apercevons que le temps de résolution du programme V1.1 est plus court que celui du programme V1, lors de la recherche de la première solution, le nombre de nœud parcouru est également plus petit. Ceci est essentiellement dû à la différence du nombre de variables et du nombre de contraintes entre ces deux programmes.

En comparant le tableau 5.4 et le tableau 5.3, nous faisons la même constatation.

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :7 311 N :2 097	T :199 449 N :60 595	T :741 533 N :175 277	T :2 138 091 N :459 859	T :483 811 N :114 913	T :3 573 173 N :719 000

TAB. 5.1 – Résultats du programme V1.1 pour la recherche de la première solution sans heuristique

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :21 281 N :3 089	T :458 404 N :89 477	T :1 648 006 N :256 004	T :4 261 360 N :660 184	T :1 075 166 N :169 924	T :6 575 560 N :1 030 060

TAB. 5.2 – Résultats du programme V1 pour la recherche de la première solution sans heuristique

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :1 660 781 N :262 064	T :2 550 679 N :490 109	T :3 172 395 N :497 777	T :4 228 825 N :660 184	T :1 074 015 N :169 924	T :6 578 679 N :1 030 060
nb Solutions	996	24	48	0	0	0

TAB. 5.3 – Résultats du programme V1 pour la recherche de toutes les solutions sans heuristique

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :889 654 N :177 838	T :1 526 004 N :338 283	T :1 775 670 N :342 059	T :2 139 006 N :459 859	T :483 987 N :114 913	T :3 573 189 N :719 000
nb Solutions	996	24	48	0	0	0

TAB. 5.4 – Résultats du programme V1.1 pour la recherche de toutes les solutions sans heuristique

Après ces comparaisons de résultats, nous constatons que le programme V1.1 est plus efficace que le programme V1.

Remarque : Le nombre de nœuds des simulations 59, 69 et 17 ne diffère pas entre la recherche de la première solution et la recherche de toutes les solutions. Cela est tout à fait normal vu que ces simulations n'ont pas de solutions dans la modélisation de "Type A" avec 3 lots.

V1.1 ("Type A") vs V2 ("Type B")

D'après les résultats des tableaux 5.5, 5.6, 5.7, 5.8, nous constatons que le programme V2 a un temps de résolution beaucoup plus petit lors de la recherche de la première et lors de la recherche de toutes les solutions. En effet en demandant ces résultats au solveur de contrainte, les pénalités appliquées par le programme V2 ne sont pas prises en compte et la résolution se ramène à la résolution d'un problème de "bin packing". De ce fait l'on pourrait penser que dans ces conditions le programme V2 doit avoir plus de solutions que le programme V1.1. Or en comparant les tableaux 5.7 et 5.8, on s'aperçoit que pour la simulation S.12, le programme V1.1 trouve 24 solutions alors que le programme V2 n'en trouve que 19. Cela peut tout à fait se produire suivant la configuration de la simulation. Car contrairement au programme V2 le programme V1.1 permet la création de plusieurs lots ayant les mêmes fenêtres temporelles.

Le programme V2 permet grâce aux pénalités de trouver la solution qui répond le mieux aux critères de la modélisation de "Type B" (quantité de produit minimum utilisé pour traiter un lot, diminution du nombre de lot...). Ce que ne fait pas le programme V1.1, pour la modélisation de "Type A", ce dernier nous donne uniquement une solution (si on lui demande de trouver la première) ou toutes les solutions. De ce fait il nous donne une solution où l'on pourra ordonnancer les parcelles à l'intérieur des lots mais pas la meilleure.

Remarque : Le programme V2 trouvera toujours une solution sauf si toute les parcelles doivent être traitées le même jour et que la durée de traitement totale dépasse les 10 heures de travail autorisées. Mais dans ce cas il est impossible à l'exploitant de traiter son exploitation si il ne travaille que 10 heures dans la journée.

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :7 311 N :2 097	T :199 449 N :60 595	T :741 533 N :175 277	T :2 138 091 N :459 859	T :483 811 N :114 913	T :3 573 173 N :719 000
H1	T :882 N :291	T :756 235 N :214 371	T :808 771 N :193 137	T :2 186 526 N :459 859	T :479 424 N :114 913	T :3 510 132 N :719 000
¬ H1	T :20 149 N :5 273	T :85 043 N :26 409	T : 36 523 N :9 484	T :2 186 616 N :459 859	T :488 077 N :114 913	T :3 385 572 N :719 000
H2	T :66 116 N :15 813	T : 140 N :49	T :42 932 N :10 662	T :2 039 944 N :459 859	T :459 235 N :114 913	T :3 280 109 N :719 000
¬ H2	T : 151 N :44	T :929 927 N :260 912	T :1 148 923 N :262 521	T :2 232 350 N :459 859	T :478 047 N :114 913	T :3 573 820 N :719 000

TAB. 5.5 – Résultats du programme V1.1 pour la recherche de la première solution

⁷uniquement pour les programmes V1 et V1.1

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :80 N :33	T :10 N :8	T :10 N :9	T :20 N :14	T :10 N :10	T :60 N :26
H1	T :20 N :33	T :10 N :8	T :20 N :9	T :10 N :13	T :10 N :10	T :10 N :28
\neg H1	T :10 N :20	T :10 N :8	T :10 N :9	T :10 N :11	T :10 N :10	T :20 N :59
H2	T :51 N :34	T :10 N :8	T :10 N :9	T :20 N :19	T :40 N :10	T :10 N :21
\neg H2	T :40 N :24	T :10 N :8	T :10 N :9	T :10 N :10	T :10 N :10	T :120 N :201

TAB. 5.6 – Résultats du programme V2 pour la recherche de la première solution

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :889 654 N :177 838	T :1 526 004 N :338 283	T :1 775 670 N :342 059	T :2 139 006 N :459 859	T :483 987 N :114 913	T :3 573 189 N :719 000
H1	T :860 592 N :177 838	T :1 514 012 N :338 283	T :1 740 093 N :342 059	T :2 186 529 N :459 859	T :479 456 N :114 913	T :3 528 134 N :719 000
\neg H1	T :768 560 N :177 838	T :1 215 123 N :338 283	T :1 706 623 N :342 059	T :2 186 643 N :459 859	T :497 008 N :114 913	T :3 385 782 N :719 000
H2	T :765 098 N :177 838	T :1 995 049 N :338 283	T :1 483 009 N :342 059	T :2 045 944 N :459 859	T :459 592 N :114 913	T :3 327 145 N :719 000
\neg H2	T :751 009 N :177 838	T :1 209 841 N :338 283	T :1 491 089 N :342 059	T :2 232 854 N :459 859	T :479 568 N :114 913	T :3 589 098 N :719 000
nb Solutions	996	24	48	0	0	0

TAB. 5.7 – Résultats du programme V1.1 pour la recherche de toutes les solutions

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :2414 N :4650	T :10 N :51	T :490 N :918	T :181 N :493	T :60 N :109	T :611 N :1763
H1	T :1553 N :4562	T :30 N :50	T :360 N :840	T :171 N :464	T :50 N :85	T :500 N :1517
\neg H1	T :2033 N :4650	T :20 N :76	T :540 N :989	T :311 N :535	T :50 N :110	T :891 N :1756
H2	T :1632 N :4555	T :20 N :70	T :351 N :853	T :250 N :512	T :60 N :85	T :731 N :1641
\neg H2	T :1712 N :4562	T :10 N :76	T :531 N :985	T :260 N :519	T :60 N :110	T :771 N :1813
nb Solutions	1371	19	263	98	24	334

TAB. 5.8 – Résultats du programme V2 pour la recherche de toutes les solutions

	S.3	S.12	S.36	S.59	S.69	S.17
SansH	T :1 782 N :3 353	T :20 N :40	T :411 N :664	T :170 N :409	T :40 N :95	T :541 N :1 455
H1	T :1 442 N :3 265	T :12 N :39	T :300 N :586	T :170 N :379	T :50 N :71	T :461 N :1 211
¬ H1	T :1 783 N :3 394	T :10 N :65	T :471 N :735	T :260 N :448	T :60 N :96	T :741 N :1 481
H2	T :1 743 N :3 349	T :100 N :59	T :401 N :599	T :300 N :433	T :60 N :71	T :641 N :1 328
¬ H2	T :1 462 N :3 283	T :10 N :65	T :451 N :731	T :240 N :431	T :50 N :96	T :751 N :1 680

TAB. 5.9 – Résultats du programme V2 pour la recherche de la solution ayant le moins de pénalité

Limites de nos heuristiques

Nous constatons en regardant le tableau 5.7 que lorsque la simulation n'a pas de solution, les heuristiques de tri n'ont aucun impact sur le temps de résolution. Ceci est normal car le solveur de contrainte va devoir parcourir le même nombre de nœuds pour s'apercevoir qu'il n'y a pas de solution, que les parcelles soient dans un ordre ou dans un autre.

En regardant le tableau 5.5 nous pourrions en déduire que nos heuristiques n'ont pas d'influence sur le temps de résolution, pour le programme V1.1. Pour la simulation S.3 c'est l'heuristique ¬H2 qui donne le meilleur résultat, l'heuristique H2 pour la simulation S.12 et l'heuristique ¬H1 pour la simulation S.36. Mais cela peut s'expliquer par le fait que dans notre modélisation la variable P_{ij} prend une valeur 0 si la parcelle i n'appartient pas au lot j et 1 sinon. Or lorsque CHOCO instancie une variable il lui attribue la plus petite valeur de son domaine, donc ici 0. Du coup la parcelle que l'on souhaiterait mettre dans un lot en premier est en fait la première parcelle que l'on ne met pas dans un lot.

D'un autre côté, en nous intéressant aux tableaux 5.6, 5.8 et 5.9, nous pouvons nous apercevoir que l'heuristique H1 procure de meilleurs résultats. Le problème d'implémentation cité précédemment n'intervient pas dans le concept de modélisation de " Type B ", car le domaine de la variable P_i représente les jours (lots) durant lesquels la parcelle i peut être traitée, du coup l'instancier à la plus petite valeur de son domaine consiste à l'affecter au lot qui représente le premier jour où elle peut être traitée. Donc, elle est mise dans un lot en priorité.

Limites de nos programmes

Que se passerait-t-il pour un plus grand nombre de parcelles et de lots ?

Le programme V1.1 va explorer un arbre de recherche contenant nbl^{nbp} feuilles pour chaque résolution. Car dans le programme V11 la disposition des fenêtres temporelles des parcelles dans le temps n'intervient pas dans la modélisation.

Alors que le programme V2 explorera dans le pire des cas un arbre de recherche contenant 5^{nbp} feuilles. Cette configuration se présenterait dans le cas où toutes les parcelles pourraient être traitées sur n'importe quel jour de l'horizon temporel. Si une telle configuration se présente il sera donc plus judicieux d'appliquer un algorithme de " Min Bin Packing " pour résoudre notre problème vu que la notion de fenêtres temporelles ne prendrait plus effet.

Le programme V1.1 va voir sa combinatoire augmenter suivant le nombre de lots voulu et le

nombre de parcelles à traiter, alors que la combinatoire programme V2 ne sera affecté que par le nombre de parcelles.

Que se passerait t-il pour un horizon temporel plus grand ?(si la fiabilité des services météorologiques le permet)

Nous venons de voir que l'arbre de recherche du programme V2 contient dans le pire des cas 5^{nbp} feuilles. Or le 5 est la taille (en jour) de notre horizon temporel, nous pouvons donc imaginer que dans le cas ou cette valeur grandirait la combinatoire de notre programme serait catastrophique. Ceci serait bien entendu vrai dans la configuration vue précédemment.

Par contre la taille de l'horizon temporel n'influe pas dans la combinatoire du programme V1.1, donc à partir d'une certaine taille pour notre horizon le programme V1.1 sera plus efficace que le programme V2.

Que se passerait t-il pour une exploitation ayant des ressources humaines plus importantes ?

Nous avons considéré que nos exploitations n'avaient qu'un seul tracteur disponible pour le traitement. Mais des exploitations ayant plusieurs tracteurs sont plus qu'envisageable. Le programme V1.1 n'a aucun moyen de spécifier le nombre de tracteurs.

Quant au programme V2 nous pouvons simuler le fait d'avoir plusieurs tracteurs en multipliant la taille de l'horizon par le nombre de tracteurs disponibles. Pour 2 tracteurs disponibles, prenons une parcelle i qui peut être traitée le jour 1 ou le jour 2, nous pourrions considérer qu'elle peut être traité le jour 1 sur le tracteur 1 ou le jour 1 sur le tracteur 2 ou le jour 2 sur le tracteur 1 ou le jour 2 sur le tracteur 2. Le domaine de la variable P_i représentant cette parcelle serait donc multiplié par le nombre de tracteur disponible. Nous obtenons donc pour le programme V2 un arbre de recherche qui contient dans le pire des cas $(NbTracteur \times 5)^{nbp}$ feuilles.

	V1.1	V2
Horizon temporel de 5 jours et 1 tracteur	nbl^{nbp}	5^{nbp}
Horizon temporel variable	nbl^{nbp}	$TailleHorizon^{nbp}$
Plusieurs tracteurs et horizon temporel variable	\emptyset	$(Nbtracteur \times TailleHorizon)^{nbp}$

TAB. 5.10 – Nombre de feuilles contenues dans l'arbre de recherche des programmes V1.1 et V2 suivant la configuration du problème

Et pour une application autre que l'agriculture ?

En regardant le tableau 5.10, nous pouvons constater qu'à partir du moment où l'application à besoin d'un horizon temporel grand, vis-à-vis de l'unité de temps utilisée, le programme V2 n'est pas du tout approprié. Par contre le programme V1.1, malgré un temps de résolution plus élevé sur notre problème, sera beaucoup plus exploitable dans un domaine industriel par exemple, où l'unité de temps serait la milliseconde et la taille de l'horizon temporel serait d'une seconde.

Dans un domaine où le nombre de lots voulus serait grand le programme V2 pourrait quant à lui facilement être utilisé.

Pour conclure

Nous venons de voir que le programme V1.1 et le programme V2 ont des performances différentes suivant la nature de l'application. De ce fait, chacun de ces programmes peut trouver sa place dans des applications de natures différentes. Malgré cela des améliorations concernant leurs implémentations sont envisageables, nous aborderons ce sujet dans la conclusion (chapitre 6) de ce mémoire.

Chapitre 6

Conclusion

6.1 Limites

Limites de la modélisation

Nous avons étudié le même outil KRONOS que lors du stage 2006 de Tu Tuitete[Tui06]. Nous avons donc rencontré les mêmes limites d'exploitation pratique de l'outil (traces de résolution pas facilement exploitables, limite des valeurs d'horloge à 16384, explosion combinatoire lorsque le nombre d'automates à combiner est important). L'approche du présent travail consistait à n'utiliser KRONOS que pour décider séparément pour chaque parcelle de la nécessité de traiter et de la plage de dates pour traiter. La technique ayant été mise en œuvre l'an passé, nous n'avons pas fait porter l'effort sur l'expérience numérique, mais sur l'analyse théorique.

En particulier, nous avons confirmé la nécessité d'employer TCTL et non CTL, afin d'être en mesure de calculer des fenêtres temporelles. Les requêtes sont faites sur un automate produit qui combine les automates liés à la règle de décision, à l'état sanitaire de la parcelle et à la météo.

Pour cette même raison de calcul de fenêtres temporelles, on est obligé de réaliser une exploration complète des chemins possibles pour l'automate produit (option allpaths de Kronos). Les mécanismes de produit "à la volée" de Kronos, au lieu de la méthode "produit puis requête", ne sont donc pas utilisés.

Nous avons également étudié les résultats d'ordonnancement par model-checking donnés dans [Abd02] (chapitre 4). Il s'est avéré que les techniques employées, et la sémantique utilisée dans la résolution, étaient fortement liées à la problématique "job-shop". De plus cette méthode est limitée par le nombre de "job" et le nombre de tâches les composant, les chiffres énoncés dans les travaux de Y. Abdeddaim[Abd02] sont 6 pour le nombre de tâches et 6 pour le nombre de jobs.

Limites des programmes d'ordonnancement

Les limites de nos programmes d'ordonnancement sont essentiellement dues à l'explosion combinatoire et à la grandeur de l'arbre de recherche. Le concept de modélisation de "type A" peut être vu comme un problème de "Bin Packing" où les objets ont une taille (durée des tâches de traitement) et un poids (quantité de produit utilisé), et où les sacs changent de taille suivant les objets mis à l'intérieur. De ce fait, la résolution des lots dans cette modélisation peut prendre beaucoup de temps. Nous avons contourné ce problème en créant le concept de modélisation de "Type B" qui peut être ramené à un problème de "Min Bin Packing" mais où tous les objets ne peuvent pas aller dans tous les sacs. Nous avons donc une vision différente mais qui répond toujours à notre problème qui est de réduire le temps de travail et le nombre de traitements effectués dans une saison.

Dans le concept de modélisation de "Type A" nous ne prenons pas en compte le fait qu'une

exploitation puisse avoir plusieurs tracteurs. Par contre ce paramètre peut être pris en compte dans le concept de modélisation de "type B", en multipliant le nombre de jours de prévoyance par le nombre de tracteurs disponibles sur l'exploitation. Cette méthode augmente bien entendu le temps de résolution.

Nous avons également incorporé directement la notion de distance entre les parcelles dans la durée de leur traitement, en ajoutant à la durée de traitement la plus grande durée de trajet séparant une parcelle des autres, dans les deux concepts de modélisation, et ainsi nous avons contourné le problème du voyageur de commerce. De ce fait, nous ne pouvons pas résoudre le problème sur des exploitations où les parcelles seraient très éloignées les unes des autres, car l'approximation en max de distance serait trop sous-optimale.

Concernant le programme linéaire en nombres entiers nous n'avons pas eu le temps de lancer des simulations avec, nous ne sommes donc pas en mesure de trouver ces limites. Nous pouvons néanmoins dire qu'il est basé sur le même principe que le concept de modélisation de "Type A" auquel nous avons ajouté le problème du voyageur de commerce. On pourrait donc prévoir un temps de résolution important.

6.2 Perspectives

A l'heure actuelle les perspectives s'orientent vers la création d'un programme d'ordonnement moins coûteux en terme de temps. Une amélioration du temps de résolution de nos programmes est envisageable, en utilisant des contraintes globales (une contrainte globale est une contrainte vérifiant la consistance de toutes les variables qu'elle affecte dans un même test). Dans les programmes V1 et V1.1 il faudrait remplacer les contraintes " IFTHEM " par des contraintes " IMPLIES " qui ont un pouvoir de propagation plus important, du fait de leur symétrie. De même, on pourrait utiliser les contraintes MIN et MAX pour le calcul des fenêtres temporelles d'un lot, ce qui n'est pas le cas actuellement à cause d'un problème d'instanciation des valeurs. En effet ces contraintes agissent sur les bornes des domaines des variables en modifiant leurs valeurs mais elles ne créent pas l'égalité entre la valeur min (resp la valeur max) d'un ensemble de variables et la variable à instancier. Grâce à la contrainte " element " (nth dans CHOCO) nous pourrions intégrer le problème de voyageur de commerce dans les programmes V1, V1.1 et V2.

Nous devons prouver que toute solution optimale de notre problème est solution du programme linéaire en nombres entiers. Ainsi le programme linéaire issu de la relaxation des contraintes d'intégrité est une borne inférieure de la solution optimale. Ainsi nous pourrions proposer un algorithme d'approximation avec une garantie de performance non triviale à un facteur constant. En clair, nous voulons savoir si notre problème est APX (au sens de la théorie de l'approximation).

6.3 Conclusion

Lors de ce stage nous avons dans un premier temps vérifié les principes de modélisation mis en place par Tu Tuitete lors de son stage en 2006[Tui06]. Nous avons apporté ici les preuves de la validité de son modèle, notamment sur le choix de la logique employée et de l'utilisation de l'outil KRONOS. Nous avons approfondi nos connaissances sur le fonctionnement du model-checking de la logique TCTL sur les automates temporisés, en donnant la construction des automates des régions. Nous nous sommes ensuite penchés sur le problème d'ordonnement en mettant en place une stratégie de création de lots au moyen de deux formalismes que sont la programmation par contrainte et la programmation linéaire. Nous avons également fait des simulations pour pouvoir en

interpréter les résultats et essayer d'en dériver des propriétés.

Malheureusement le stage ne dure que 6 mois ce qui fait que nous n'avons pas eu le temps de pousser toutes nos recherches là où nous l'aurions souhaité. Il aurait été sûrement plus efficace de lancer nos recherches seulement sur la partie ordonnancement de notre problème qui n'avait presque pas été traité dans Tu Tuitete[Tui06].

Dans la dernière partie du stage, nous valoriserons nos connaissances en model-checking en contribuant à la version final de [ON07].

Ainsi, après construction de nos exploitations sous forme d'automates temporisés, les diverses requêtes faites à l'outil de model-checking KRONOS, suivies de l'exécution de l'un de nos deux programmes nous pouvons donner une réponse au problème :

Quels seraient les lots de travail et leurs positions dans le temps permettant l'application d'une règle de gestion dans une exploitation donnée ?

Bibliographie

- [Abd02] Y. Abdeddaïm. *Modélisation et Résolution de Problème d'Ordonnancement à l'aide d'Automates Temporisés*. PhD thesis, Ecole Doctorale Electronique, Elestrotechnique, Automatique, Télécommunication, Signal, 2002.
- [AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990.
- [AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.
- [CD95] S. Tripakis S. Yovine C. Daws, A. Olivero. The tool KRONOS. In *Hybrid Systems III : Verification and Control*, volume 1066, pages 208–219, Rutgers University, New Brunswick, NJ, USA, 22–25 October 1995. Springer.
- [Cha03] F. Charfi. Une approche d'interfaçage de cod à uppaal pour la spécification et la vérification des systèmes temps réel. Master's thesis, Faculté des Sciences de Tunis, 2003.
- [FC03] O. H.Roux F. Cassez. Traduction structurelle des réseaux de petri temporels en automates temporisés, 2003.
- [GB02] A.D. Kim G.larsen P. Pettersson W. Yi G. Behrmann, J. Bengtsson. *UPPAAL Implementation Secrets*. Departement of Information Technology, Uppsala University, Sweden Basic Research in Computer Science, Aalborg University, Denmark, 2002.
- [Gou99] A. Gouin. *Contribution à la commande de systèmes à événements discrets temporisés : synthèse de superviseur dans le cadre de modèle automate*. PhD thesis, Ecole Doctorale Sciences pour l'Ingénieur de Nantes, 1999.
- [Kat98] J.P. Katoen. Concepts, algorithms and tools for model checking. Lecture Notes, University of Twente, september 1998.
- [Kat03] J.P. Katoen. Principles of model checking. Lecture Notes, University of Twente, 2003.
- [Lar05] F. Laroussinie. *Model checking temporisé — Algorithmes efficaces et complexité*. Mémoire d'habilitation, Université Paris 7, Paris, France, December 2005.
- [ON07] B. Léger A. Hélias R. Giroudeau O. Naud, T. Tuitete. Systèmes réactifs pour modéliser la décision en production agricole. MSR2007, soumis en mars 2007.
- [Tui06] T. Tuitete. Ordonnancement de tâches phytosanitaires vinicoles. Master's thesis, Université Montpellier II, 2006.
- [WY94] M. Daniels W. Yi, P. Pettersson. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7th Int. Conf. on Formal Description Techniques*, pages 223–238. North-Holland, 1994.
- [Yov93] S. Yovine. *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, 1993.