Analyse de l'autostabilisation de l'unisson synchrone via la satisfiabilité propositionnelle

Asma Khoualdia, Sami Cherif, Stéphane Devismes, Léo Robert

Laboratoire MIS UR 4290, Université de Picardie Jules Verne, Amiens, France

asma.khoualdia@etud.u-picardie.fr,{sami.cherif;stephane.devismes;leo.robert}@u-picardie.fr

Résumé

L'unisson synchrone est un problème de synchronisation d'horloges de référence en système distribué, et notamment en autostabilisation. Cet article explore l'autostabilisation d'un algorithme d'unisson synchrone en utilisant une approche basée sur la satisfiabilité propositionnelle. Une formulation logique de l'algorithme est proposée, intégrant l'unicité des valeurs d'horloge des nœuds, leurs mises à jour en fonction de la valeur minimale des horloges des nœuds voisins, ainsi que la vérification de la convergence ou de la divergence. Afin d'optimiser la modélisation, des contraintes additionnelles sur les configurations initiales sont introduites pour limiter les cas redondants et non pertinents à analyser.

Mots-clés

Autostabilisation, unisson synchrone, satisfiabilité

Abstract

Synchronous unison is a benchmark clock synchronization problem in distributed computing, and especially in self-stabilization. This paper explores the self-stabilization of a synchronous unison algorithm using a propositional satisfiability-based approach. A logical formulation of the algorithm is proposed, incorporating the uniqueness of node clock values, their updates based on the minimum clock value at neighboring nodes, and the verification of the convergence or divergence. To optimize the modeling, additional constraints are introduced to reduce redundant and irrelevant cases to be analyzed.

Keywords

Self-stabilization, synchronous unison, satisfiability

1 Introduction

L'*autostabilisation* a été définie par Dijkstra [10] dans le contexte des *systèmes distribués*. Un système distribué est un ensemble d'entités de calcul, ici appelées *processus*, qui sont à la fois *autonomes* et *interconnectées* entre elles. Le but de ces processus est de coopérer afin de résoudre une tâche globale au système. Un algorithme est dit *autostabilisant* si, quel que soit l'état initial du système sur lequel il est déployé, il garantit le retour en temps fini, et sans intervention extérieure, à un état dit *légitime* à partir duquel la spécification du système est réalisée. Bien que la notion de panne ne soit pas explicite dans la définition des systèmes autostabilisants, la motivation première de cette approche est pourtant la *tolérance aux pannes*. En effet, après un nombre fini de *pannes transitoires* [1], l'état d'un système distribué peut être quelconque et ainsi ne plus vérifier sa spécification. Un algorithme autostabilisant garantit alors que le système récupère de telles fautes en temps fini.

La conception d'algorithmes autostabilisants peut sembler complexe au premier abord en raison de l'accès limité des processus à l'état global du système. Pourtant, nombre de ces algorithmes se révèlent étonnamment élégants et parfois même plus simples que leurs pendants non stabilisants. Cependant, démontrer formellement l'autostabilisation d'un algorithme s'avère souvent ardu et sujet aux erreurs. Cela est principalement dû à la nature combinatoire du problème : la convergence vers un état légitime doit être démontrée à partir de *tout* état du système. Dans cet optique, nous nous intéressons ici à un problème simple : l'*unisson synchrone*. On suppose un système synchrone où les processus sont anonymes et arbitrairement connectés entre-eux. Chaque processus dispose d'une horloge locale dont les valeurs entières varient entre 0 à $m-1$ où m est appelée la *période*. À partir d'un état où l'horloge de chaque processus a une valeur quelconque de $\{0,\ldots,m-1\}$, le but est de faire converger le système vers un état à partir duquel toutes les horloges sont identiques et s'incrémentent (modulo m) à chaque pas de calcul.

Arora et al. ont proposé une solution simple à ce problème [2]. L'autostabilisation de leur algorithme est démontrée pour tout $m \geq \max(2, 2\mathcal{D}-1)$ où $\mathcal{D}$ est le diamètre du réseau [1, 2]. Cependant, l'optimalité de cette borne reste une question ouverte. Cette question est partiellement résolue dans [1] où des *états de divergence*, c'est-à-dire, des états à partir desquels des exécutions ne stabilisent jamais, sont exhibés pour toutes périodes *paires* inférieures strictement à la borne ; le cas impair restant non résolu à ce jour. Nous proposons ici d'étudier comment la satisfiabilité propositionnelle peut aider à traiter cette question, et plus généralement comment elle peut aider à vérifier l'autostabilisation d'algorithmes distribués.

1. C'est-à-dire, des perturbations temporaires de certains composants du systèmes (liens ou processus).

La *satisfiabilité* propositionnelle (SAT) est un problème fondamental en logique et en informatique théorique, consistant à déterminer si une formule booléenne peut être satisfaite par une assignation de valeurs. Ce problème, qui a été le premier démontré NP-complet [8], suscite un intérêt croissant car les solveurs SAT modernes parviennent à traiter efficacement de grandes instances issues d'applications concrètes, malgré la difficulté théorique établie du problème. En plus de son rôle clé en vérification de logiciels, d'intelligence artificielle et de cryptographie, SAT se situe à l'intersection de plusieurs disciplines telles que la logique, la complexité, la programmation par contraintes ou encore la théorie des graphes [3].

Dans cet article, nous proposons une approche formelle basée sur SAT pour analyser rigoureusement le comportement de l'unisson synchrone d'Arora et al. [2]. Les règles de mise à jour des horloges sont traduites en contraintes logiques sous forme normale conjonctive (FNC), permettant l'utilisation de solveurs SAT pour vérifier si la synchronisation est atteinte. Cette méthodologie offre un cadre systématique pour étudier la stabilisation de l'algorithme dans différentes topologies et configurations initiales et avec différentes périodes. Pour optimiser cette analyse, des techniques spécifiques sont appliquées pour réduire les redondances des configurations initiales, ce qui permet d'accélérer l'évaluation du modèle. Cette approche, bien qu'axée sur l'unisson, fournit des garanties formelles concernant les scénarios de synchronisation, tout en ouvrant des perspectives d'optimisation dans des systèmes distribués plus complexes.

Le reste de cet article est organisé comme suit. Dans la section 2, nous présentons l'algorithme d'unisson synchrone et nous introduisons la satisfiabilité propositionnelle. La section 3 est consacrée à la modélisation formelle de l'algorithme de l'unisson synchrone d'Arora et al. [2], où nous détaillons les variables et notations, ainsi que la formulation en forme normale conjonctive, avec une attention portée sur le maintien de l'unicité des horloges, la mise à jour des horloges et la modélisation la propriété d'autostabilisation, mais aussi la modélisation de contraintes additionnelles sur les configurations initiales dans le but d'accélérer la résolution du problème. Enfin, la section 4 présente une évaluation expérimentale des résultats obtenus. Nous concluons et discutons les perspectives de nos travaux en section 5.

2 Préliminaires

2.1 Unisson synchrone

Nous considérons un système distribué de n processus interconnectés. Le réseau d'interconnexion est modélisé par un graphe non-orienté connexe $G = (V, E)$ où V est un ensemble de n sommets représentant les processus et E un ensemble d'arêtes représentant les liens de communication bidirectionnels entre les processus. L'ensemble des voisins d'un processus p sera noté $N(p)$. L'unisson synchrone d'Arora et al. [2] est décrit dans le *modèle à états*, un modèle à mémoires localement partagées où chaque nœud peut directement lire son état et ceux de ses voisins, mais peut uniquement modifier son propre état.

Étant donné un paramètre entier m commun à tous les processus et appelé *période*, chaque processus p dispose d'une seule variable $p.c \in \{0, \ldots, m-1\}$, qui représente son *horloge*. La *configuration* du système est ainsi modélisée par le vecteur associant à chaque processus la valeur de son horloge. L'ensemble des configurations possibles du système sera noté $\Gamma_{n,m}$. L'algorithme local de chaque processus p est décrit par une unique règle de la forme $garde \rightarrow action$. La garde de la règle de p est un prédicat sur l'état de p et ceux de ses voisins. La partie action est une affection de l'horloge de p. Dans une configuration donnée, la règle d'un processus est dite activable si sa garde est vraie. Une configuration où aucune règle n'est activable est dite *terminale*.

Le système est supposé synchrone : tant que le système n'est pas dans une configuration terminale, des itérations sont exécutées comme suit. Tous les processus dont la règle est activable dans la configuration courante γ_i exécutent simultanément l'action de leur règle, générant ainsi une nouvelle configuration γ_{i+1}, et ainsi de suite. Une exécution de l'algorithme est donc une suite de configurations $\gamma_0, \ldots, \gamma_i, \ldots$ telle que pour tout $i > 0$, γ_i est obtenu à partir de γ_{i-1} en une itération synchrone de l'algorithme. Le but de l'algorithme est d'atteindre une configuration où toutes les horloges se sont synchronisée sur une valeur identique pour n'importe quelle configuration initiale possible.

Pour ce faire, à chaque itération, chaque processus p calcule la valeur d'horloge minimale $c_{\min}$ dans son voisinage clos (c'est-à-dire, $N(p) \cup \{p\}$), puis modifie, si nécessaire, son horloge $p.c$ à $(c_{\min} + 1) \bmod m$ (cf. Algorithme 1). De proche en proche, les valeurs les plus petites se propagent dans le réseau. Cette convergence peut uniquement être perturbée par l'apparition de 0 chez certains processus après que tout leur voisinage clos ait atteint la valeur maximale $m-1$. Arora et al. ont démontré que lorsque m est choisi suffisamment grand (c'est-à-dire, $m \geq \max(2, 2\mathcal{D}-1)$ où $\mathcal{D}$ est le diamètre de G), ce souci de réinitialisation locale ne peut pas empêcher le système de converger puisque, quelle que soit la configuration initiale $\gamma_0 \in \Gamma_{n,m}$, il existe nécessairement une valeur $t \geq 0$ telle que toutes les horloges ont la même valeur dans γ_t suite à une exécution de l'algorithme.

Dans le cas où m est choisi trop petit, l'exécution peut

Algorithme 1 Unisson synchrone pour un processus p

Entrées :
$N(p)$: l'ensemble des voisins de p
m : un entier positif, la période

Variable :
$p.c \in \{0, .., m-1\}$, l'horloge de p

Macro :
$New(p) = (\min(\{q.c : q \in N(p)\} \cup \{p.c\}) + 1) \bmod m$

Règle :
$p.c \neq New(p) \rightarrow p.c := New(p)$

ne jamais converger. Dans ce cas, il existe un cycle entre des configurations *illégitimes* (des configurations contenant au moins deux valeurs différentes). Or, le nombre total de configurations possibles est m^n. Donc, si après $m^n - 1$ itérations, le système est toujours dans une configuration illégitime, on peut affirmer que l'exécution ne convergera jamais. Ainsi, il est suffisant d'observer les $t_f = m^n$ premières configurations d'une exécution pour pouvoir décider si cette exécution converge ou diverge. Cependant, ce nombre est très grand et peut entraîner une modélisation exponentielle. On pourra alors choisir une valeur plus raisonnable pour t_f, quitte à ne pas pouvoir décider dans certains cas.

Exemple 1. *Considérons un réseau de processus G=($\{p_0, p_1, .., p_5\}, E$) dont la topologie est en anneau. Posons $m = 5$ et supposons la configuration initiale γ_0 donnée dans la Figure 1. À partir de γ_0, on obtient le préfixe d'exécution $\gamma_0, \gamma_1\gamma_2, \gamma_3$ donné dans la Table 1. L'algorithme d'unisson atteint donc une configuration légitime après 3 itérations.*

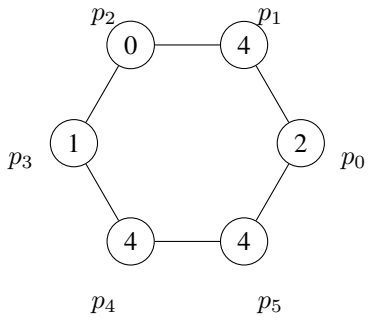


FIGURE 1 – Configuration initiale γ_0 dans un anneau de 6 nœuds avec $m = 5$.

	p_0	p_1	p_2	p_3	p_4	p_5
γ_0	2	4	0	1	4	4
γ_1	3	1	1	1	2	3
γ_2	2	2	2	2	2	3
γ_3	3	3	3	3	3	3

TABLE 1 – Préfixe de 4 configurations obtenu après 3 itérations depuis γ_0.

Dans l'exemple précédent, une configuration légitime, où les horloges sont synchronisées est atteinte par l'algorithme. Cependant, lorsque m est mal choisi, il peut exister des configurations à partir desquelles les horloges ne se synchronisent jamais. On dira dans ce cas que le système *diverge*. Les propriétés de convergence et de divergence, formellement définies ci-dessous, mettent en évidence l'importance d'étudier le comportement des algorithmes distribués, et particulièrement l'unisson synchrone, et les paramètres influençant la synchronisation dans son contexte [9, 11].

Définition 1 (Convergence)**.** *Étant donné une période m, l'algorithme d'unisson synchrone exécuté sur un graphe G de n sommets converge lorsque pour toute configuration initiale $\gamma_0 \in \Gamma_{n,m}$, on atteint en un nombre fini d'itérations une configuration $\gamma_t \in \Gamma_{n,m}$ où toutes les valeurs d'horloge des sommets sont identiques.*

Définition 2 (Divergence)**.** *Pour une période donnée m, l'algorithme d'unisson synchrone exécuté sur un graphe G de n sommets diverge lorsqu'il existe une configuration initiale $\gamma_0 \in \Gamma_{n,m}$ à partir de laquelle les horloges ne se synchronisent jamais : il existe une infinité de configurations atteintes contenant au moins deux valeurs d'horloge distinctes.*

Exemple 2. *Considérons un réseau de $n = 3$ processus dont la topologie est une chaîne. Posons $m = 2$ et considérons la configuration initiale γ_0 donnée dans la Figure 2. À partir de cette configuration initiale, on obtient le préfixe d'exécution décrit dans la Table 2. La divergence est donc atteinte dès la deuxième itération puisqu'on retrouve une configuration identique à γ_0.*

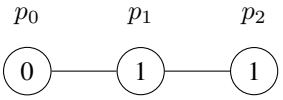


FIGURE 2 – Configuration initiale γ_0 dans une chaîne de 3 nœuds avec $m = 2$.

	p_0	p_1	p_2
γ_0	0	1	1
γ_1	1	1	0
γ_2	0	1	1

TABLE 2 – Deux itérations de l'unisson synchrone sur une chaîne de 3 nœuds avec $m = 2$.

Notre analyse va porter sur l'impact de la topologie du graphe et de la période m sur la synchronisation, en se concentrant sur des topologies simples comme les lignes, les anneaux et les étoiles. Le théorème suivant démontré dans [1] fournit notamment une limite générale intéressante pour le nombre maximum d'itérations dans un graphe connecté.

Théorème 1. *L'algorithme d'unisson synchrone est autostabilisant dans tout graphe connecté de diamètre $\mathcal{D}$ pour $m \geq \max\{2, 2\mathcal{D} - 1\}$. Son temps de stabilisation est alors d'au plus $3\mathcal{D} - 2$ itérations.*

2.2 Satisfiabilité propositionnelle

Soit X un ensemble des variables propositionnelles. Un littéral est une variable $x \in X$ ou sa négation $\overline{x}$. Une clause est une disjonction de littéraux. Une formule en Forme Normale Conjonctive (FNC) est une conjonction de clauses. Une affectation $I : X \to \{vrai, faux\}$ associe à

chaque variable une valeur booléenne et peut être représentée comme un ensemble de littéraux. Un littéral est satisfait par une affectation I si $l \in I$, sinon il est falsifié par I. Une clause est satisfaite par une affectation I si au moins un de ses littéraux est satisfait par I, sinon elle est falsifiée par I. Une formule FNC est satisfiable s'il existe une affectation qui satisfait toutes ses clauses, sinon elle est insatisfiable. Le problème de satisfiabilité (SAT) consiste à déterminer si une formule FNC donnée est satisfiable.

Bien que SAT soit NP-complet [8], les solveurs modernes basés sur l'algorithme CDCL (Conflict Driven Clause Learning) [12] sont efficaces et parviennent à résoudre des instances impliquant un grand nombre de variables et de clauses. En effet, au delà de l'apprentissage de clauses et du retour arrière non chronologiques, ces solveurs intègrent des mécanismes puissants comme, par exemple, les structures paresseuses, des heuristiques de branchement dédiées ou encore les redémarrages [6]. Par ailleurs le problème SAT est largement utilisé pour modéliser et résoudre de nombreux problèmes combinatoires complexes issus de divers domaines, notamment en vérification formelle [4, 7] mais aussi en cryptographie, en bio-informatique et en planification [6].

Dans la suite, on s'intéresse à modélisation du comportement de l'algorithme d'unisson synchrone, appliqué à un graphe arbitrairement connecté. L'objectif est de caractériser les conditions de convergence ou de divergence de l'algorithme. Cela implique la modélisation des états des horloges des nœuds du réseau à l'aide de formules logiques, qui doivent être exprimées de manière à pouvoir être résolues efficacement par un solveur SAT.

3 Modélisation formelle de l'unisson synchrone

Dans cette section, nous nous intéressons à la modélisation formelle du comportement de l'unisson synchrone à travers la satisfiabilité propositionnelle. Nous commençons par introduire les variables du modèle, puis nous définissons les contraintes qui encadrent l'exécution de l'algorithme. Ces contraintes doivent non seulement représenter fidèlement l'évolution du système, mais aussi permettre de détecter sa convergence ou sa divergence. On s'intéressera également à l'introduction de contraintes permettant d'optimiser le modèle en éliminant des configurations initiales redondantes.

3.1 Variables et Notations

On rappelle que le réseau est modélisé par un graphe non-orienté connexe $G = (V, E)$ où $V = \{0, \ldots, n-1\}$ est un ensemble de n sommets représentant les processus. On notera $M = \{0, \ldots, m-1\}$ l'ensemble des valeurs d'horloges possibles, où m est la période. Nous nous intéressons aux t_f premières configurations d'une exécution quelconque de l'algorithme sur G. Elles seront indicées de 0 à $t_f - 1$. L'ensemble des indices sera noté $T = \{0, \ldots, t_f - 1\}$. Pour modéliser l'exécution de l'algorithme, il est aussi nécessaire de raisonner sur l'état des horloges des processus dans les différentes configurations. Pour cela, on introduit les variables d'horloge $h_{p,t,v}$, pour tout $(p, t, v) \in V \times T \times M$. Chaque variable $h_{p,t,v}$ indiquera si l'horloge du processus p est égale à la valeur v dans la configuration t. Cela s'avère suffisant pour modéliser l'exécution et pour détecter la convergence. Cependant, la modélisation de la divergence nécessite d'autres variables, notamment pour pouvoir représenter la présence d'un cycle entre configurations illégitimes pendant l'exécution et pour simplifier la transformation des contraintes en forme normale conjonctive. L'ensemble des variables qui seront utilisées est donné ci-dessous :

- **Variables d'horloge :** $h_{p,t,v}$ où $p \in V, t \in T, v \in M$. Ces variables sont affectées à $vrai$ si le processus p a la valeur d'horloge v dans la configuration t et à faux sinon.
- **Variables de cycle :** c_t où $t \in T$. Ces variables sont affectées à $vrai$ si la configuration $t \in T$ et la configuration initiale sont identiques.
- **Variables de similarité d'horloge :** $s_{p,t,v}$ où $p \in V, t \in T$ et $v \in M$. Ces variables seront affectées à $vrai$ si le processus p a la même valeur d'horloge dans la configuration $t \in T$ et la configuration initiale.

3.2 Modélisation de l'exécution de l'unisson

Dans un premier temps, nous nous intéressons à la formalisation des contraintes modélisant une exécution valide de l'unisson synchrone. Plus précisément, nous définissons les contraintes d'unicité des horloges ainsi que les règles de mise à jour assurant la propagation correcte des valeurs d'horloge au fil des itérations.

3.2.1 Unicité des horloges

Pour garantir une exécution valide de l'algorithme, il est nécessaire de maintenir l'unicité de l'horloge de chaque processus à chaque itération. Précisément, il faut veiller à ce que chaque processus possède une et une seule valeur d'horloge à chaque configuration de l'algorithme. Cette contrainte peut être modélisée comme suit :

$$\bigwedge_{\substack{(v,v') \in M^2 \\ v < v'}} (\overline{h_{i,t,v}} \vee \overline{h_{p,t,v'}}) \quad \forall p \in V, \forall t \in T \tag{1a}$$

$$\bigvee_{v \in M} h_{p,t,v} \quad \forall p \in V, \forall t \in T \tag{1b}$$

La contrainte (1a) garantit qu'un processus ne peut pas posséder simultanément deux valeurs d'horloge différentes dans une configuration donnée tandis que la contrainte (1b) assure que chaque processus possède au moins une valeur d'horloge valide à chaque configuration. Le nombre de clauses induit par ces contraintes est en $O(n * t_f * m^2)$.

3.2.2 Mise à jour des horloges

Dans le cadre de l'algorithme de l'unisson synchrone, les horloges des processus doivent être mises à jour, à chaque itération, en fonction des valeurs d'horloge de leurs voisins immédiats. Plus précisément, la nouvelle valeur d'horloge du processus p est déterminée en prenant le minimum des

valeurs d'horloge de ses voisins, ainsi que la valeur d'horloge qu'il détenait à l'étape précédente. Cette valeur sera ensuite incrémentée de 1 modulo m. Pour simplifier les notations, on dénote $\tilde{N}(p) = N(p) \cup \{p\}$ le *voisinage clos* du processus $p \in V$. La contrainte de mise à jour des horloges est alors modélisée comme suit :

$$\bigwedge_{\substack{(v_{p'})_{p' \in \tilde{N}(p)} \in M^{d_p} \\ v' = (min_{p' \in \tilde{N}(p)} v_{p'} + 1) \bmod m}} \left(\bigvee_{p' \in \tilde{N}(p)} \overline{h_{p',t,v_{p'}}} \right) \vee h_{p,t+1,v'}$$
$$\forall t \in T \setminus \{t_f - 1\}, \forall p \in V \quad (2)$$

En effet, pour chaque assignation possible $(v_{p'})_{p' \in \tilde{N}(p)}$ des valeurs d'horloges du voisinage clos d'un processus p, on veut s'assurer que la valeur minimale incrémentée de 1 modulo m, dénotée v', soit propagée au processus p à la configuration suivante, c'est à dire $\bigwedge_{p' \in \tilde{N}(p)} h_{p',t,v_{p'}} \rightarrow h_{p,t+1,v'}$. Cette dernière implication met parfaitement en évidence le mécanisme de mise à jour des horloges dans le cadre de l'unisson synchrone et est donc réécrite sous forme normale conjonctive dans la contrainte (2), en rajoutant les itérations nécessaires sur les assignations possibles et le calcul de la nouvelle valeur de l'horloge au niveau de la conjonction. Pour fournir une analyse fine de la complexité de l'encodage, on note $d_{\max} = \max_{p \in V} |N(p)|$ le degré maximal du graphe. Ainsi, il est clair que le nombre de clauses serait borné par $O(n * t_f * \times m^{d_{\max}+1})$. Il semblerait donc que la complexité de notre modèle soit fortement corrélée au degré maximal dans le graphe et qu'il soit donc plus adapté aux graphes pour lesquels cette valeur reste relativement petite. En particulier, pour les graphes en anneau ou en chaîne la complexité sera en $O(n * t_f * \times m^3)$ puisque, pour ces topologies, on a $d_{\max} = 2$.

3.3 Modélisation de la convergence

La convergence se produit lorsque les valeurs d'horloge des processus atteignent en un nombre fini d'itérations une configuration *légitime*, où toutes les valeurs d'horloge sont identiques pour l'ensemble des processus. Un algorithme est dit convergent si, à partir de n'importe quelle configuration initiale, une telle configuration légitime est toujours atteinte en un nombre fini d'itérations. Pour démontrer cette propriété, nous allons raisonner par l'absurde en supposant que la convergence n'est jamais atteinte à partir d'une certain configuration initiale. Autrement dit, il existe (au moins) une configuration initiale, à partir de laquelle aucune configuration légitime ne peut être obtenue. Étant donné qu'une configuration légitime engendre nécessairement une autre configuration légitime après une itération de l'algorithme, il suffit alors de montrer que la dernière configuration $t_f - 1$ n'est pas légitime pour une certaine configuration initiale. Cette propriété est ainsi traduite par la contrainte suivante :

$$\bigwedge_{v \in M} \bigvee_{p \in V} \overline{h_{p,t_f-1,v}} \quad (3)$$

Ainsi, il est clair que la formule CNF ϕ obtenue grâce aux contraintes (1), (2) et (3) est insatisfiable si et seulement si l'algorithme d'unisson synchrone converge. En effet, si ϕ est satisfiable, le modèle fourni permet d'exhiber une configuration initiale $\lambda_0 \in \Gamma_{n,m}$ déduite des variables d'horloge $h_{p,0,v}$ affectées à $vrai$ pour $p \in V$ et $v \in M$ à partir de laquelle l'algorithme ne converge pas. Plus spécifiquement, après une exécution valide de l'algorithme d'unisson à partir de λ_0 (assurée par les contraintes (1) et (2)), on arrive toujours à exhiber, au niveau de la dernière configuration t_f et pour toute valeur d'horloge possible $v \in M$, un processus $p \in V$ dont la valeur d'horloge ne prend pas la valeur v dans t_f (assuré par la contrainte (3)). En terme de complexité, la contrainte (3) nécessitant seulement $O(m)$ clauses, notre modèle pour la convergence nécessite seulement $O(n \times t_f \times m)$ variables avec un nombre de clauses borné en $O(n * t_f * m^{d_{max}+1})$ où d_{max} est le degré maximal du graphe.

3.4 Modélisation de la divergence

La divergence implique l'existence de deux configurations illégitimes identiques dans l'exécution. En effet, l'ensemble des configurations légitimes est trivialement clos et l'algorithme étant synchrone et déterministe, un tel cycle se répète à l'infini. Soit deux configurations $(t_1, t_2) \in T^2$ tels que $t_1 < t_2$, où les valeurs d'horloge des processus en t_2 sont identiques à celles en t_1. On appellera une telle occurrence un *cycle* au niveau des configurations (t_1, t_2). Pour établir la divergence, il faut donc modéliser l'existence d'un cycle au niveau de deux configurations illégitimes possibles atteintes par l'algorithme. Pour simplifier la modélisation, on considère l'existence de cycle par rapport à la configuration initiale, c'est à dire $t_1 = 0$. En effet, même si $t_1 > 0$, il est toujours possible de ramener l'analyse à la configuration initiale en prenant la même configuration comme état initial du système (*n.b.*, un algorithme autostabilisant doit converger depuis n'importe quelle configuration initiale). De plus, pour garantir que les configurations du cycle soient illégitimes, il suffit simplement de vérifier que la configuration initiale est illégitime. On rappelle que les variables c_t où $t \in T$, définies auparavant, représentent l'existence de cycle à la configuration t. Ainsi l'existence d'un cycle entre configurations illégitimes peut se traduire par les contraintes suivantes :

$$\bigvee_{\substack{t \in T \\ 0 < t}} c_t \quad (4a)$$

$$\bigwedge_{v \in M} \bigvee_{p \in V} \overline{h_{p,0,v}} \quad (4b)$$

Clairement, la contrainte (4a) modélise l'existence d'un cycle $(0, t)$ où $t \in T$; tandis que la contrainte (4b) garantit que la configuration 0, et donc toutes les configurations du cycles, sont illégitimes. Pour instaurer le sens sémantique de variables de cycle, il faut garantir que, si ses variables sont affectées à $vrai$, alors il y a un cycle $(0, t)$, c'est à dire que les valeurs d'horloge dans la configuration t

sont identiques à celles dans la configuration initiale. Dans ce contexte, les variables de similarité d'horloge $s_{p,t,v}$ où $p \in V$, $t \in T$ et $v \in M$ peuvent jouer le rôle intermédiaire de garantir une même valeur d'horloge dans la configuration $t \in T$ et la configuration initiale. Ainsi, on rajoute les contraintes suivantes :

$$\overline{c_t} \vee \bigvee_{v \in M} s_{p,t,v} \quad \forall t \in T, \forall p \in V \tag{5a}$$

$$(\overline{s_{p,t,v}} \vee h_{p,0,v}) \wedge (\overline{s_{p,t,v}} \vee h_{p,t,v}) \quad \forall t \in T, \forall p \in V, \forall v \in M \tag{5b}$$

La contrainte (5a) assure que la configuration $t \in T$ correspondante a les mêmes valeurs d'horloge que la configuration 0 ou, plus formellement, que $c_t \rightarrow \bigwedge_{p \in V} \bigvee_{v \in M} s_{p,t,v}$. La forme normale présentée dans la contrainte (5a) peut donc être simplement obtenue par la réécriture clausale de l'implication ainsi que la distributivité des disjonctions sur les conjonctions logiques. La contrainte (5b) est nécessaire pour instaurer le sens sémantique des variables $s_{p,t,v}$ qui sont affectées à $vrai$ quand le processus $p \in V$ a la même valeur d'horloge $v \in V$ aux configurations 0 et t. Plus formellement, on a $s_{p,t,v} \rightarrow h_{p,0,v} \wedge h_{p,t,v}$ et sa forme normale présentée dans la contrainte (5b) peut également être déduite par réécriture de l'implication et distributivité de la disjonction.
Ainsi, il est clair que la formule CNF ψ obtenue par les contraintes (1), (2), (4) et (5) est satisfiable si et seulement si l'algorithme d'unisson diverge sur le réseau donné en entrée avec la période m. En effet, un modèle de ψ permet d'exhiber une configuration initiale $\lambda_0 \in \Gamma_{n,m}$ à partir de la quelle une exécution valide de l'unisson synchrone génère un cycle $(0, t)$ de configurations illégitimes, donné par une variable de cycle affectée à $vrai$ au niveau de la contrainte clausale (4a). De plus, en terme de complexité de l'encodage pour la divergence, il est clair que les contraintes (4) et (5) introduisent un nombre de clauses borné en $O(n \times t_f \times m)$ et, par conséquent, que le modèle a une complexité similaire au cas de la convergence, principalement régie par la contrainte (2), c'est à dire en $O(n * t_f * m^{d_{max}+1})$ où d_{max} est le degré maximal du graphe. Cependant, le modèle nécessite un nombre plus grand de variables, mais qui reste borné en $O(n \times t_f \times m)$.

3.5 Élimination de configurations initiales

Dans cette section, nous cherchons à réduire le nombre de configurations initiales à traiter en supprimant les configurations redondantes. En d'autres termes, il s'agit d'éliminer les configurations pour lesquelles le comportement de l'algorithme est identique à celui d'autres configurations déjà prises en compte. Cette approche permet d'optimiser l'exécution de l'algorithme en évitant de traiter des situations qui, bien que différentes dans leur représentation, ne conduisent pas à des résultats différents.

3.5.1 Ordre lexicographique

Dans de le cadre de l'algorithme d'unisson, l'évolution des valeurs d'horloge des processus dépend exclusivement de leur voisinage local et, plus particulièrement, de leur voisinage clos. En particulier, deux processus possédant le même ensemble de voisins dans la topologie étudiée évoluent de manière identique, à permutation près de leurs valeurs d'horloge initiales. Cette observation permet d'exploiter une symétrie structurelle afin de réduire l'espace des configurations à considérer en instaurant un ordre lexicographique particulier sur les valeurs initiales d'horloge pour des processus ayant un voisinage identique dans la topologie. Pour tirer parti de cette symétrie et éviter d'examiner plusieurs configurations équivalentes, nous introduisons des contraintes imposant un ordre lexicographique sur les configurations. Plus précisément, seules les configurations minimales selon cet ordre sont considérées, ce qui permet d'éliminer les configurations redondantes et de réduire l'espace de recherche. Dans notre cas, on va introduire des contraintes interdisant les configurations lexicographiquement plus grandes sur les voisinages locaux identiques que celles sélectionnées. Cette contrainte, qu'on dénotera OL, peut être formulée comme suit :

$$\bigwedge_{\substack{(p,p') \in V^2 \\ p<p', \\ N(p)=N(p')}} \bigwedge_{\substack{(v,v') \in M^2 \\ v'<v}} (\overline{h_{p,0,v}} \vee \overline{h_{p',0,v'}}) \tag{6}$$

Ainsi, si deux processus $(p, p') \in V^2$ tels que $p < p'$ ont un voisinage identique, la contrainte (6) garantit que p' ne puisse pas prendre une valeur d'horloge initiale plus petite que celle de p. Cette contrainte introduit un nombre de clauses borné en $O(n^2 \times m^2)$ et ne peut être appliquée que dans des topologies où certains processus possèdent un voisinage identique, comme dans la topologie en étoile, où un processus central (hub) est connecté à plusieurs processus périphériques (voir Figure 4), ou encore dans les graphes complets.

3.5.2 Élimination des rotations

Comme étayé précédemment, lorsqu'une transformation d'une configuration initiale ne modifie pas le comportement global de l'algorithme, il est possible d'éliminer les configurations redondantes afin d'optimiser la recherche et l'analyse des cas pertinents. Un cas particulièrement intéressant concerne les graphes, tels que les anneaux, où une même configuration peut être exprimée sous plusieurs formes équivalentes via une simple rotation des indices des processus. Dans ces structures, toute configuration initiale peut être transformée en une autre configuration par un décalage uniforme des valeurs d'horloge. Dès lors que la rotation ne modifie ni les relations entre voisins ni la dynamique de l'algorithme, on peut restreindre l'étude à un sous-ensemble représentatif des configurations possibles.
Afin d'imposer cette restriction, on peut introduire une contrainte garantissant que, parmi toutes les configurations équivalentes, seules celles où un processus spécifique détient la plus faible valeur d'horloge sont considérées. En particulier, dans un graphe en anneau de n processus, on considère une configuration où le premier processus possède la valeur d'horloge la plus faible parmi les valeurs

d'horloge des autres processus dans la même configuration. Puisque la rotation, qu'elle soit dans le sens direct ou indirect, conserve le même comportement de l'algorithme d'unisson synchrone, on éliminera toute autre configuration initiale qui est image de cette configuration afin d'éviter l'analyse de cas redondants, comme démontré dans la proposition 1, avec $\lambda[p]$ qui désigne la valeur d'horloge de $p \in V$ dans la configuration $\lambda \in \Gamma_{n,m}$. Cette contrainte, qu'on dénotera ER, peut donc être exprimée comme suit sur toute topologie en anneau :

$$\bigwedge_{\substack{(v,v')\in M^2 \\ v<v'}} \bigwedge_{p\in V\setminus\{0\}} (\overline{x_{0,0,v'}} \vee \overline{x_{p,0,v}}) \tag{7}$$

Proposition 1. *Soit $G = (V, E)$ un graphe en anneau composé de n processus et m une période. Soit $ER = \{\lambda \in \Gamma_{n,m} \mid \exists p \in V \setminus \{0\}, \lambda[p] < \lambda[0]\}$. Alors, pour toute configuration initiale $\lambda_0 \in ER$, il existe une configuration $\lambda'_0 \in \Gamma_{n,m} \setminus ER$ à partir de laquelle une exécution de l'unisson synchrone produit le même comportement de convergence ou de divergence.*

Démonstration. Sans perte de généralité, on suppose que $V = \{0, \ldots, n-1\}$. Soit $\lambda_0 \in ER$ et $p_{\min} = \mathsf{argmin}_{p\in V}\lambda_0[p]$. Nous définissons la configuration λ'_0 telle que :

$$\lambda'_0[p] = \begin{cases} \lambda[p - p_{\min}] & \text{pour } p_{\min} \leq p \leq n-1, \\ \lambda[n - p_{\min} + p] & \text{pour } 0 \leq p < p_{\min}. \end{cases}$$

Clairement, λ'_0 est obtenue par une rotation de λ_0 qui ramène la valeur la plus petite au processus 0. Donc, λ'_0 a le même comportement que λ, à rotation près, suite à une exécution de l'algorithme d'unisson synchrone puisque, sous rotation, les relations entre voisins sont préservées et chaque processus applique exactement les mêmes règles d'évolution que dans la configuration originale. Par ailleurs, puisque $\lambda'_0[0] = \lambda_0[p_{\min}] = \min_{p\in V} \lambda[p]$, il n'existe pas de processus $p \in V \setminus \{0\}$ tel que $\lambda[p] < \lambda[0]$ et donc λ'_0 appartient bien au noyau $\Gamma_{n,m} \setminus ER$. □

La contrainte (7) introduit un nombre de clauses borné en $O(n \times m^2)$ et peut donc être appliquée sur les graphes dont la topologie permet de faire des rotations particulières en préservant les relations entre voisins comme la topologie en anneau ou encore la topologie en étoile en appliquant la contrainte uniquement sur les processus périphériques.

4 Évaluation expérimentale

4.1 Protocole expérimental

Nous avons encodé nos modèles en Python en utilisant la bibliothèque PySAT [2]. Nous avons généré une instance pour chaque type de topologie (étoiles, chaînes et anneaux), chaque nombre de nœuds et chaque période. Nous avons résolu ces instances en utilisant le solveur Cadical [5]. Les tests ont été réalisés sur la plateforme MatriCS [3], sur une machine équipée d'un processeur Intel Core i7 cadencé à 3,80 GHz, sous Ubuntu 22.04. Un temps limite de 3600 s a été considéré pour chaque instance.

Nous avons généré un total de 720 instances réparties comme suit. Pour chaque type de topologie parmi les anneaux, les chaînes et les étoiles, on prend des tailles de graphes n allant de 3 jusqu'à 10 noeuds, et on fait varier les périodes m de 2 à 10. Ainsi, on obtient 72 formules de convergence (resp. divergence) avec le *modèle initial* (INI) incluant les contraintes (1), (2) et (3) (resp. (1), (2), (4) et (5)) pour chaque topologie. Par ailleurs, pour les topologies permettant des contraintes d'élimination de configurations initiales, on génère également les instances correspondantes. Ainsi, on a généré 144 instances de convergence et de divergence pour la topologie en anneau (resp. étoile) avec la contrainte d'élimination des rotation (resp. d'ordre lexicographique). Enfin, on fixe $t_f = 100$ pour toutes les instances.

4.2 Analyse de la convergence et la divergence de l'unisson synchrone

Les résultats présentés dans la Figure 3 illustrent le comportement de l'algorithme d'unisson synchrone en termes de convergence et de divergence sur les différents types de topologies (anneaux, chaînes et étoiles) en fonction de la taille n et de la période m. Pour les anneaux et les chaînes, toutes les instances ont pu être résolues, révélant une convergence alignée avec le Théorème 1 lorsque $m \geq max\{2, 2\mathcal{D} - 1\}$. On rappelle que ces topologies présentent des diamètres qui évoluent en fonction de la taille des graphes avec $\mathcal{D} = \lfloor n/2 \rfloor$ pour les anneaux et $\mathcal{D} = n-1$ pour les chaînes.

En revanche, pour les graphes en étoile, certaines instances n'ont pas été résolues dans la limite d'une heure. Ceci est traduit par la complexité accrue du modèle dans le cadre de la topologie en étoile, à cause du noeud central dont le voisinage inclut tous les noeuds périphériques. Les résultats expérimentaux en terme de performance confirmeront également cette observation dans la section 4.3. Cependant, il est intéressant de noter que nos résultats ont montré que la borne proposée dans Théorème 1 est précise dans le cas des étoiles. En effet, ce théorème indique que la convergence est garantie dans les étoiles (de diamètre 2) à partir d'une période m supérieure ou égale à 3 ($\max(2, 2 \times 2 - 1) = 3$). Or, nos résultats exhibent un cas de divergence avec $m = 2$ dans les graphes en étoiles, indépendamment de leurs tailles. Ainsi, l'algorithme d'unisson synchrone converge dans une étoile si et seulement si $m \geq 3$. Nous démontrons que notre contre-exemple pour $m = 2$ est général dans les étoiles dans la proposition suivante.

Proposition 2. *L'algorithme d'unisson synchrone diverge pour tout graphe en étoile, quel que soit son nombre de nœuds $n \geq 3$, avec la période $m = 2$.*

2. https://pysathq.github.io/

3. https://www.matrics.u-picardie.fr/catalogue-de-services/compte-matrics/

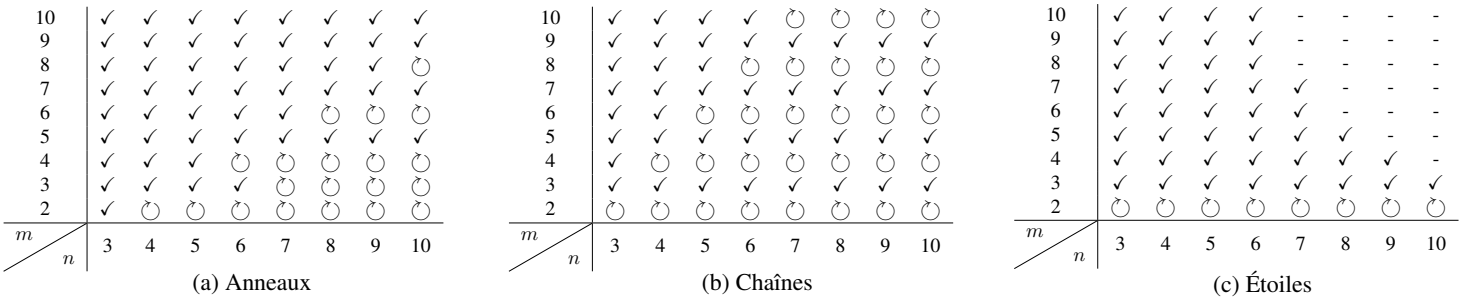


FIGURE 3 – Résultats de convergence et de divergence de l'unisson synchrone sur les anneaux, chaînes et étoiles, pour différentes tailles de graphe n et périodes m. Les signes (✓), (↻) et (−) indiquent respectivement que l'algorithme converge, diverge ou que l'instance n'a pas été résolue dans le temps alloué.

Démonstration. Nous pouvons toujours exhiber une configuration initiale $\gamma_0 \in \Gamma_{n,m}$ qui entraîne la divergence de l'unisson synchrone. Une telle configuration est illustrée dans la Figure 1, où la valeur initiale de l'horloge du nœud central est fixée à 1, tandis que les valeurs des horloges des nœuds périphériques prennent au moins deux valeurs initiales distinctes. Sans perte de généralité, nous supposons que tous les nœuds ayant une valeur 0 sont situés à gauche du nœud central, tandis que les nœuds ayant une valeur d'horloge égale à 1 sont situés à sa droite.

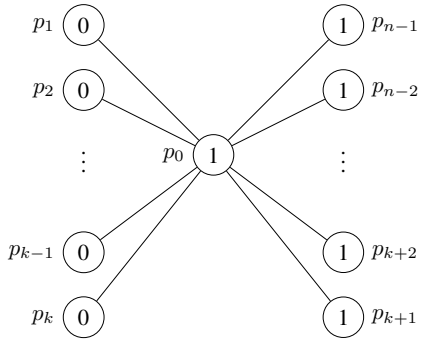


FIGURE 4 – Configuration initiale assurant la divergence de l'unisson synchrone sur un graphe en étoile avec n noeuds et en fixant la période à $m = 2$.

Lors de l'exécution de l'algorithme d'unisson synchrone sur un tel graphe avec la configuration initiale correspondante, le nœud central mettra à jour son horloge à 1, les nœuds périphériques situés à gauche mettront également à jour leurs horloges à 1, tandis que les nœuds périphériques situés à droite mettront à jour leurs horloges à 0. À l'itération suivante, un cycle de configurations illégitimes est alors atteint, car nous revenons à la configuration non convergente γ_0. Étant donné que ce comportement se maintient pour un graphe en étoile de toute taille, nous concluons que l'unisson synchrone diverge sur de tels graphes, indépendemment de leurs tailles, lorsque $m = 2$.

□

4.3 Analyse des performances

Dans cette section, nous analysons les performances des modèles sur différentes topologies en termes de temps moyen de résolution par taille de graphe. Les résultats obtenus pour les graphes en anneau, en chaîne et en étoile sont présentés dans le Tableau 3, en fonction de la taille du graphe, de la propriété analysée et du modèle considéré. Les résultats indiquent que le modèle initial (INI) parvient à résoudre l'ensemble des instances générées pour les graphes en anneau et en chaîne. En revanche, ce même modèle rencontre des difficultés croissantes avec la taille du graphe pour la topologie en étoile. Il parvient ainsi à résoudre 51 (resp. 50) instances sur 72 pour la vérification de convergence (resp. divergence) sur les étoiles. Ce comportement n'est pas surprenant, car cette topologie présente un degré maximal $d_{\max} = n - 1$, ce qui entraîne une explosion combinatoire du nombre de clauses générées par la contrainte (2). En effet, celui-ci croît selon une complexité de l'ordre de $O(n \times t_f \times m^n)$, ce qui explique la difficulté du modèle à gérer les grandes instances.

Les Figures 5 et 6 illustrent l'évolution du temps de résolution des instances générées pour la vérification de la divergence, en fonction de la taille du graphe n et de la période m, respectivement pour les topologies en anneau et en étoile. Ces courbes mettent en évidence une augmentation progressive du temps de résolution à mesure que ces deux paramètres augmentent. On observe plus particulièrement que, dans le cas des graphes en étoile, cette croissance est plus marquée que pour les graphes en anneau. Cette tendance, qui reste similaire pour la convergence ainsi que pour les graphes en chaîne, corrobore nos observations antérieures et confirme la complexité accrue du modèle pour certaines topologies.

Dans la suite, nous étudions les formulations augmentées par les contraintes d'élimination des configurations initiales. Comme l'illustre le Tableau 3, la comparaison avec le modèle initiale met en évidence la pertinence d'introduire ces contraintes, en particulier pour l'analyse de la divergence. Les résultats montrent une réduction significative du temps de résolution pour les topologies en anneau et en étoile grâce aux contraintes d'élimination de rotation (ER) et d'ordre lexicographique (OL), respectivement. Plus précisément, pour les graphes en anneau, le modèle ER atteint les meilleures performances globales, avec un temps total de résolution de 18.1s pour la convergence et 57.61s pour la divergence, sur l'ensemble des tailles de graphes testées.

n	Anneaux				Chaînes		Étoiles			
	CONV		DIV		CONV	DIV	CONV		DIV	
	INI	ER	INI	ER	INI	INI	INI	OL	INI	**OL**
3	**0.01**	**0.01**	0.60	**0.31**	0.01	0.36	**0.006**	**0.006**	0.26	**0.16**
4	0.18	**0.11**	1.96	**0.60**	0.10	1.4	**0.15**	0.33	3.10	**1.29**
5	0.53	**0.52**	4.88	**1.58**	0.34	3.73	**5.04**	5.78	46.10	**11.90**
6	**1.13**	**1.13**	11.07	**3.17**	0.53	7.03	**123.25**	176.65	483.09 [8]	**211.90**
7	**1.71**	2.07	19.72	**5.45**	2.11	8.01	102.23 [6]	**89.43 [6]**	591.61 [6]	**92.33 [6]**
8	3.19	**2.72**	35.30	**9.72**	3.20	14.78	**48.76** [4]	52.00 [4]	351.62 [4]	**39.56 [4]**
9	4.54	**3.93**	64.80	**16.88**	4.57	21.39	**44.89** [3]	56.34 [3]	188.65 [3]	**15.61 [3]**
10	8.11	**7.61**	59.59	**19.90**	7.94	28.07	7.58 [2]	**0.81 [2]**	28.73 [2]	**1.63 [2]**

TABLE 3 – Temps de résolution en secondes pour les topologies en anneau, chaîne et étoile. Pour chaque topologie, les temps moyens de résolution pour la détection de la convergence (CONV) et la divergence (DIV) sur toutes les périodes pour une taille de graphe donnée (n) sont reportés. Les résultats des modèles optimisés sont également reportés pour les topologies permettant des contraintes supplémentaires (ER pour les anneaux et OL pour les étoiles) et les meilleurs résultats entre les modèles initial et optimisé sont marqué en gras. Pour une taille n donnée avec des instances ayant atteint le temps limite, le nombre d'instances résolues est indiqué entre '[]'.

Comparé au modèle initial, ER permet ainsi de réduire le temps de résolution total de 6.701% pour la convergence et de 70.892% pour la divergence. Ces résultats sont aussi mis en évidence dans la Figure 5, démontrant ainsi l'impact positif de cette optimisation sur l'analyse du comportement de l'unisson synchrone. Le graphe en étoile révèle une dynamique similaire puisque le modèle OL démontre une amélioration significative du temps de divergence avec une valeur totale de 374.38s, soit une réduction de 77.858% par rapport au modèle initial (1 693.16s). Ces observations sont aussi confirmé par la Figure 5 et mettent en avant la pertinence de cette optimisation dans des structures où les interactions sont fortement centralisées autour d'un nœud pivot. Toutefois, pour la convergence, le modèle OL reste légèrement en retrait par rapport au modèle ER, ce qui suggère que son efficacité se manifeste principalement dans la vérification de la divergence.

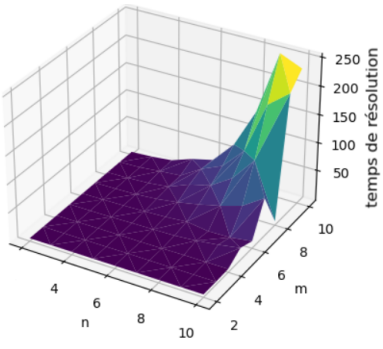
Pour résumer, les résultats montrent que, même sans optimisation spécifique, les modèles sont capables d'analyser le comportement de l'unisson synchrone, notamment pour les graphes en chaîne qui présentent déjà des temps de résolution compétitifs. Les modèles optimisés, en particulier ER, se révèlent être les plus efficaces pour réduire l'efficacité du modèle dans les graphes en anneau, tandis que OL se distingue par une performance meilleure lors de la vérification de la divergence dans les graphes en étoile, d'autant plus que ce modèle a permis de résoudre une instance de plus par rapport au modèle initial.

5 Conclusion

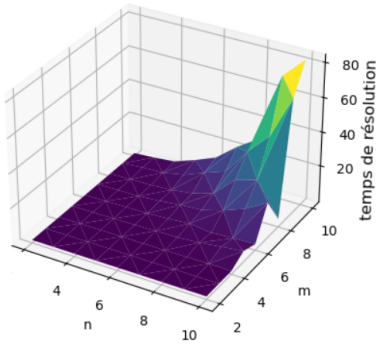
Dans cet article, nous avons présenté une approche formelle basée sur la satisfiabilité propositionnelle (SAT) pour analyser le comportement de un algorithme autostabilisant d'unisson synchrone. Précisément, on s'est intéressé à la détection des cas où cet algorithme converge ou diverge. En représentant les états du problème et son exécution en contraintes logiques, nous avons montré comment les solveurs SAT peuvent être utilisés pour prouver l'autostabilisation d'un système distribué ou, dans le cas contraire, pour produire un contre-exemple. Ce cadre a permis d'étudier rigoureusement l'autostabilisation d'un algorithme d'unisson synchrone dans des topologies variées, en prenant en compte différentes périodes. En outre, nous avons optimisé l'analyse en proposant des contraintes spécifiques pour éliminer les configurations initiales redondantes, ce qui a permis d'accélérer l'évaluation du modèle et de garantir des résultats plus efficaces. Nos modèles ont également permis d'exhiber des comportements particuliers, notamment la divergence dans le cas des étoiles quand $m = 2$.

Plusieurs pistes de recherche restent ouvertes. Tout d'abord, l'extension de notre analyse à d'autres types de graphes, tels que les grilles, les tores ou les arbres binaires, permettrait d'examiner plus en détail l'impact d'autres topologies distribuées classiques sur l'autostabilisation de l'algorithme. Ces structures, particulièrement pertinentes pour des applications concrètes, offrent une grande diversité de configurations qui méritent une investigation approfondie afin d'affiner notre compréhension des dynamiques de convergence et de divergence. On pourra aussi affiner le choix du nombre maximal de configurations autorisées par rapport à la topologie étudiée.

Il serait également pertinent d'étudier d'autres formulations plus compactes de la contrainte de mise à jour des horloges afin de réduire la complexité du modèle. De plus,pour la contrainte de convergence, il serait possible d'intégrer une analyse plus fine en instaurant des contraintes redondantes sur l'ensemble des configurations au lieu de se focaliser uniquement sur la vérification de la synchronisation au niveau de la dernière. Une telle approche permettrait alors de détecter plus rapidement les configurations légitimes. Une autre perspective intéressante porte sur la prise en compte des symétries pour la détection des cycle divergents. Ces perspectives ouvrent la voie à une meilleure compréhension des dynamiques de synchronisation dans les systèmes distribués complexes, et à la mise au point de nouvelles méthodes plus efficaces et optimisées pour traiter un éventail plus large de cas d'applications.

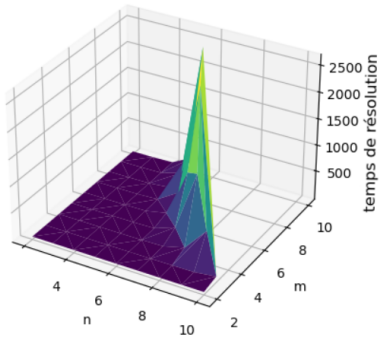


(a) INI

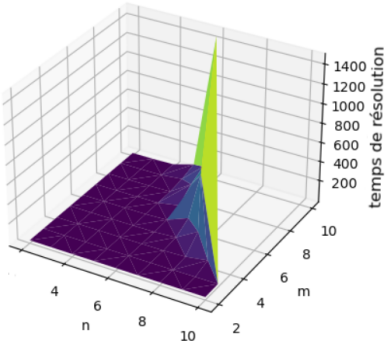


(b) ER

FIGURE 5 – Temps de résolution en secondes pour la divergence en fonction du nombre de nœuds n et de la période m pour le graphe des anneaux avec le modèle initial (INI) et augmenté par l'élimination des rotations (ER)



(a) INI



(b) OL

FIGURE 6 – Temps de résolution en secondes pour la divergence en fonction du nombre de nœuds n et de la période m pour les graphes en étoile avec le modèle initial (INI) et augmenté par l'ordre lexicographique (OL)

Remerciements

Ce travail est partiellement soutenu par le projet ANR-24-CE23-6126 (BforSAT) financé par l'agence nationale de recherche. Il a bénéficié d'un accès aux ressources HPC de la « Plateforme MatriCS » de l'Université de Picardie Jules Verne, qui est cofinancée par l'Union Européenne avec le Fonds Européen de Développement Régional (FEDER) et le Conseil Régional des Hauts-De-France entre autres.

Références

[1] Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2019.

[2] Anish Arora, Shlomi Dolev, and Mohamed G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1 :11–18, 1991.

[3] Armin Biere. Handbook of satisfiability. In *Frontiers in Artificial Intelligence and Applications*, pages 75–98. IOS Press, 2009.

[4] Armin Biere et al. Symbolic model checking without bdds. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.

[5] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *International Conference on Computer Aided Verification, CAV*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024.

[6] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of Satisfiability : Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

[7] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

[8] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[9] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison (extended abstract). In *The 12th International Conference on Distributed Computing Systems (ICDCS)*, pages 486–493. IEEE Computer Society, 1992.

[10] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.

[11] Danny Dolev and Dahlia Malkhi. Consensus : Perspectives and challenges. In *Proceedings of the Tenth International Workshop on Distributed Algorithms (WDAG)*, pages 1–12, 1995.

[12] João P. Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, 1999.