Optimisation de l'insertion de contre-mesures pour la sécurité des circuits intégrés

Jonathan Fontaine

Thèse présentée pour l'obtention du grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Spécialité

Sciences et technologies de l'information et de la communication

École doctorale

Informatique, Télécommunication et Électronique Paris (ED130)

Optimisation de l'insertion de contre-mesures pour la sécurité des circuits intégrés

---

Jonathan Fontaine

Soutenue publiquement le : *24 mai 2024*

Devant un jury composé de :

Philippe Coussy, Professeur, Université de Bretagne Sud, LAB-STICC	*Rapporteur*
André Rossi, Professeur, Université Paris-Dauphine, LAMSADE	*Rapporteur*
Sophie Dupuis, Maîtresse de Conférences, Université de Montpellier, LIRMM	*Examinatrice*
Ozgur Sinanoglu, Professeur, New York University Abu Dhabi	*Examinateur*
Alix Munier, Professeure, Sorbonne Université, LIP6	*Présidente de jury*
Gabriel Gouvine, Ingénieur, Auto-Entrepreneur	*Invité*
Lilia Zaourar, Ingénieur de Recherche, CEA-LIST, DSCIN	*Encadrante*
Roselyne Chotin, Maître de Conférences, Sorbonne Université, LIP6	*Directrice de thèse*

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de cette thèse. Je remercie l'équipe d'encadrement de m'avoir proposé un sujet à mi-chemin entre l'optimisation et l'électronique, un domaine que j'ai dû apprendre à maîtriser à vos côtés. Votre soutien et votre expertise ont été inestimables tout au long de ce parcours. Je remercie particulièrement ma directrice de thèse, Roselyne Chotin, qui s'est battue avec détermination pour mener cette thèse jusqu'à la soutenance. Je remercie Philippe Coussy et André Rossi d'avoir accepté de rapporter ma thèse.

Je remercie mes anciens collègues du CEA et du LIP6, chercheurs et doctorants, pour tous ces échanges, scientifiques et non scientifiques, autour de cafés. Je tiens également à remercier mes camarades d'ACTIF pour m'avoir ouvert à d'autres horizons scientifiques, pour m'avoir permis d'échanger sur nos thèses ainsi que pour avoir proposé quelques moments de décompression, mérités et savourés.

Je remercie mes amis, collègues ou non, scientifiques ou non, pour m'avoir accompagné de près ou de loin durant ce projet. Je les remercie de m'avoir fait sortir de ma bulle, et de m'avoir proposé des activités qui me permettaient de respirer.

Enfin, je tiens tout particulièrement à remercier Julien Rodriguez et Valentin Gilbert pour leur support face aux doutes, leur soutien inconditionnel, et surtout leur magnifique camaraderie qui m'a permis de finir cette épreuve.

Abstract

Over the last 75 years, the electronics industry has experienced a spectacular evolution, moving from manual design to an automated industry. This industrialization has led to increased complexity in circuits, requiring specialization in tasks during the design of electronic circuits. Various companies around the world have emerged to perform these tasks, with varying levels of trust assigned. From a designer's perspective, these actors pose several threats, such as the insertion of malicious functionalities, intellectual property theft, or circuit counterfeiting. These threats impact the economy of the semiconductor industry, amounting to billions of dollars in losses annually.

One way to combat these threats is to lock the circuit with a key, preventing it from functioning correctly if the right key is not present. *Logic locking* is a method that involves logically locking a circuit using key gates and the corresponding digital key. Several implementations of *logic locking* have been developed. In these works, we focus on *Strong Logic Locking*. It locks the circuit by connecting $XOR/XNOR$ gates to the digital key, inserted in circuit signals. Each insertion position has a different impact on security, which is the possibility of recovering the digital key. However, adding logic gates in a circuit increases power consumption, the circuit's area, and decreases performance. *Strong logic locking* aims to maximize the security of the lock by identifying positions that enhance security, regardless of the resulting impact.

In this thesis, we seek to optimize security while considering the impact on circuit performance. We propose a new approach to solving *strong logic locking*. We start by formulating our security problem based on mathematical models that include security for optimally inserting key gates in the circuit. This formulation calculates the cliques of a subgraph representing the insertion positions. We establish a *branch and bound* solving algorithm for our problem and evaluate it. We then present a second mathematical models representing the impact on the delay from inserting key gates in the circuit. Finaly, we propose strategies to optimize security while limiting the impact on circuit performance. Our tools are integrated into the design flow, allowing us to validate them with numerical results obtained on circuits used by the electronic community.

Résumé

En 75 ans d'existence, l'industrie de l'électronique a connu une évolution spectaculaire, passant d'une conception manuelle à une industrie automatisée. Cette industrialisation a entraîné une complexification des circuits, nécessitant une spécialisation des tâches lors de la conception d'un circuit électronique. Différents acteurs à travers le monde sont apparus pour réaliser ces tâches, avec différents niveaux de confiance accordés. Du point de vue d'un concepteur, ces acteurs apportent plusieurs menaces, telles que l'insertion de fonctionnalités malveillantes, le vol de propriété intellectuelle ou la contrefaçon de circuits. Ces menaces impactent l'économie de l'industrie des semiconducteurs et représentent plusieurs milliards de dollars de pertes par an.

Une façon de lutter contre ces menaces est de verrouiller le circuit avec une clé, l'empêchant de fonctionner correctement si la bonne clé n'est pas présente. Le *logic locking* est une méthode consistant à verrouiller logiquement un circuit à l'aide de portes clés et de la clé numérique correspondante. Plusieurs implémentations de *logic locking* ont été réalisées. Dans ces travaux, nous retenons le *Strong Logic Locking*. Elle verrouille le circuit en reliant des portes $XOR/XNOR$ à la clé numérique, insérée sur des signaux du circuit. Chaque position d'insertion a une incidence différente sur la sécurité, qui est la possibilité de retrouver la clé numérique. Toutefois, ajouter des portes logiques dans un circuit augmente la consommation électrique, la surface du circuit et diminue les performances. Le *strong logic locking* vise à maximiser la sécurité du verrouillage en cherchant les positions qui maximisent la sécurité, sans se soucier de l'impact généré.

Dans cette thèse, nous cherchons à optimiser la sécurité tout en prenant en compte l'impact sur les performances du circuit. Nous proposons une nouvelle approche de résolution du *strong logic locking*. Nous commençons par formuler notre problème de sécurité en nous basant sur des modèles mathématiques incluant la sécurité pour insérer de manière optimale les portes clés dans le circuit. Cette formulation calcule les cliques d'un sous-graphe représentant les positions d'insertion. Nous proposons un algorithme de résolution *branch and bound* pour notre problème que nous évaluons. Nous présentons ensuite d'autres modèles mathématiques représentant l'impact sur le délai de l'insertion de portes clés dans le circuit. Puis nous développons des stratégies pour optimiser la sécurité tout en limitant l'impact sur les performances du circuit. Nos outils sont intégrés dans le flot de conception, ce qui nous permet de les valider avec des résultats numériques obtenus sur des circuits utilisés par la communauté électronique.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Sommaire

1.1 Contexte

1.1.1 Émergence de l'électronique

En 75 ans d'existence [BB48], l'industrie de l'électronique a connu une évolution spectaculaire. De la conception manuelle des composants électroniques à une industrie massive, elle est devenue un pilier de l'économie moderne. Cette industrialisation a entraîné une complexification des circuits, permettant d'augmenter les capacités de calcul, une demande insatiable dans nos sociétés modernes. Cette évolution a également entraîné une augmentation des coûts de production, ainsi qu'une spécialisation des tâches lors de la conception d'un circuit électronique. Différents acteurs sont apparus pour réaliser ces tâches afin de limiter les coûts et de mutualiser les investissements.

Aujourd'hui, l'électronique est omniprésente dans nos sociétés, utilisée dans tous les domaines, des moins cruciaux, comme le divertissement, aux plus critiques, comme la santé ou le transport. Elle couvre une multitude d'applications, allant de dispositifs portables, dans un *pacemaker*, à des super-calculateurs nécessitants des bâtiments dédiés. Cette adoption généralisée crée des besoins spécifiques, conduisant à une spécialisation croissante de chaque acteur de l'industrie, de la conception à la fabrication.

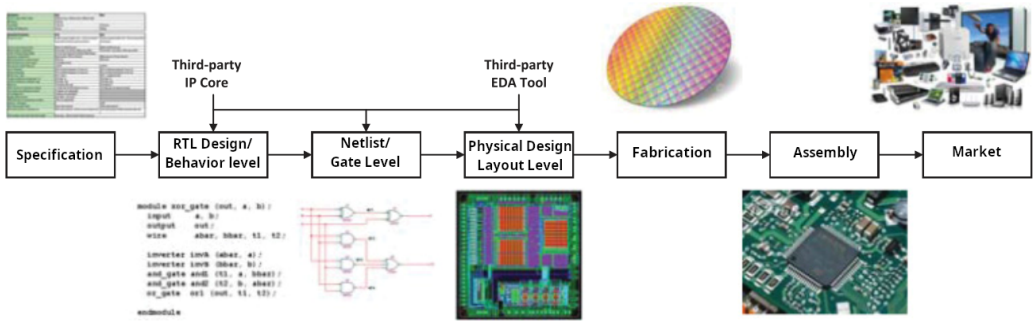


Figure 1.1. : Flot de conception d'un circuit intégré

La conception d'un circuit est une suite de tâches complexes, qui est divisée en trois grandes étapes, résumée dans la Figure 1.1. La première étape est le *High Level Synthesis (HLS)*. Elle transforme une description algorithmique du circuit (C/C++ par exemple) en une description comportementale avec des registres (*Register Transfer Level : RTL*), souvent décrite en Verilog / VHDL. La deuxième étape s'appelle la synthèse logique (*logic synthesis*). Elle consiste à transformer une description *RTL* d'un circuit en une description *netlist*, qui est une représentation des interconnexions entre les blocs logiques. Ces blocs peuvent être des portes logiques ou des structures plus complexes. La troisième étape est le placement-routage (*place and route* ou *physical synthesis*). Elle consiste à déterminer la position des composants et des fils qui les connecte à partir d'une description *netlist* du circuit et produit la description des masques (plan) qui sert à la fabrication. Le placement a pour objectif de positionner les composants électroniques en minimisant la distance entre chaque composant connecté, sous contrainte d'une limite de densité. Enfin, le routage consiste à construire des connexions entre les composants sur des pistes métalliques. Ces trois étapes peuvent être réalisées manuellement, mais la complexité et la taille des circuits actuels imposent d'utiliser des outils automatisés appelés *Electronic Design Automation (EDA)*. Une fois ces étapes réalisées, les masques (*layout*) sont envoyés dans une fonderie, qui va fabriquer des disques de plusieurs circuits, appelés *wafer*. Enfin ces *wafers* sont assemblés pour produire le circuit intégré afin d'être vendu sur le marché.

Lors de la conception d'un circuit, certains acteurs proposent des briques technologiques, appelées *Intellectual Property (IP)*. Ces briques technologiques permettent de partager des composants communs au fonctionnement d'un circuit, tels que des blocs de calcul ou des blocs mémoires, cela permet aux concepteurs de se concentrer sur les parties spécifiques du circuit et d'assembler les IPs en fonction des besoins.

Les fabricants de semi-conducteurs, socles de l'industrie, se sont développés et ont évolué vers des installations hautement spécialisées, appelées fonderies, pour répondre à la demande croissante de puces électroniques. Une fonderie moderne nécessite plusieurs milliards de dollars d'investissement pour s'équiper de chaînes de fabrication capables de produire des puces avec une précision nanométrique. Cet investissement restreint drastiquement le nombre d'acteurs dans ce domaine, obligeant ainsi à déléguer la fabrication à ces derniers. Parmi les plus gros acteurs, il y a *TSMC*, *Globalfoundries*, *Samsung Semiconductor* et *STMicroelectronics*.

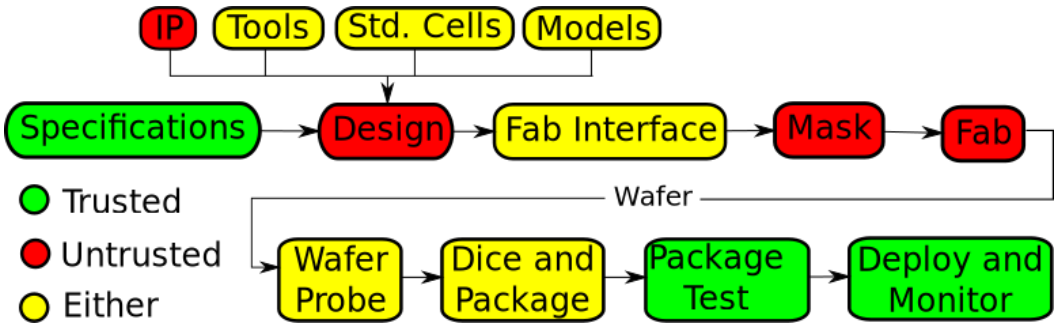


Figure 1.2. : Confiance accordée dans le flot de conception, du point de vue des concepteurs (Mentor Graphics 2015)

L'essor d'une industrie complexe, atomisée et mondialisée apporte de nouvelles problématiques. En prenant le point de vue d'un concepteur voulant produire un circuit, plusieurs menaces peuvent survenir, la Figure 1.2 montre le niveau de confiance accordé à chaque acteur. Ajouter des contre-mesures à chaque étape de la conception pour contrevenir à ces menaces à un impact non-négligeable sur le coût de fabrication et les performances du circuits. Dans cette thèse, nous nous intéressons à quatres menaces en particulier. La première est l'insertion de fonctionnalités malveillantes via des outils *EDA*. Les outils existants sont très complexes et coûteux en développement. Les outils les plus utilisés sont des outils commerciaux, avec un code fermé pour lesquels il est difficile de vérifier l'ajout de fonctionnalités cachées sur des circuits de grande taille. Des outils avec un code ouvert *(open source)* le permettent. Généralement moins puissants que leurs homologues fermés [PT13], ils apportent une fiabilité supplémentaire en contrepartie. Ces outils sont des briques essentielles pour le matériel libre *(open hardware)*. La deuxième menace est une IP vérolée. Une fonction cachée et indésirable peut être introduite dans un but malveillant, appelée Cheval de Troie Matériel (*CTM* ou *Hardware Trojan*). La troisième se produit lors de la fabrication d'un circuit. Un fabricant peut modifier le circuit pendant la fabrication pour y ajouter un CTM, un composant électronique malveillant. L'impact de ces CTMs peut aller d'un vol de données confidentielles, comme une clé cryptographique privée, en passant par l'implémentation de portes

dérobées (*backdoors*), à un dysfonctionnement volontaire du calcul, tel qu'un changement de valeurs reçues des capteurs physiques [Kar+10]. La quatrième menace est le vol de propriété intellectuelle [Cla06]. Elle peut se présenter sous trois formes, une copie d'une IP, une rétro-ingénierie *(reverse engineering)* ou une contrefaçon de circuit. En effet, le nombre de circuits créés est un accord entre le concepteur et la fonderie. Cependant, rien n'empêche alors une fonderie de surproduire en secret pour ensuite, revendre sur le marché noir ces circuits fonctionnels aux prix de fabrications, sans rémunérer les travaux de conception. Une entreprise de conception qui vend une *IP* peut se la faire copier par un client qui peut l'utiliser ou la revendre en dehors du contrat négocier, et de toute compensation financière prévue. Pour copier l'architecture, elle peut accéder aux sources de l'IP ou la reproduire avec de la rétro-ingénierie. La majeure partie du coût final d'un circuit est due à la recherche et développement (R&D) nécessaire pour le réaliser. La perte économique liée au vol d'IP se chiffrerait à plusieurs milliards de dollars par an [RSK14a]. Il est nécessaire de concevoir des méthodes pour contrer ces différentes menaces provenant de la chaîne de production.

1.1.2 Chevaux de Troie matériels

Les CTMs sont des composants électroniques malicieux et furtifs. Ils sont composés de deux parties : le *Trigger* et le *Payload*. Le *Payload* est la charge virale, c'est-à-dire la fonction logique que l'attaquant souhaite introduire. Le *Trigger* est le contrôleur logique qui active ou désactive le *Payload*. Ce dispositif est indispensable pour assurer la furtivité du CTM. Son activation dépend de différents signaux du circuit, appelés *Trigger Inputs*. Sans lui, un circuit vérolé pourrait être considéré comme défaillant lors des phases de tests post-production. La Figure 1.3 est un exemple de CTM qui inverse un signal cible lorsque les conditions du *Trigger* sont réunies.

De nombreuses méthodes, avec leurs avantages et inconvénients, ont été développées pour lutter contre les CTMs, regroupées en deux catégories : détection et prévention. Nous commençons par présenter les méthodes de détections.

Une première approche consiste à comparer le circuit avant et après fabrication en appliquant une méthode de *reverse engineering* [BFS16]. Cette méthode permet de récupérer la *netlist* du circuit fabriqué et de la comparer avec la *netlist* originale. Néanmoins, cette méthode présente un désavantage, elle détruit le circuit observé. Ceci permet de confirmer que ce circuit n'a pas été vérolé, mais aucune certitude ne peut être apportée pour les autres circuits.

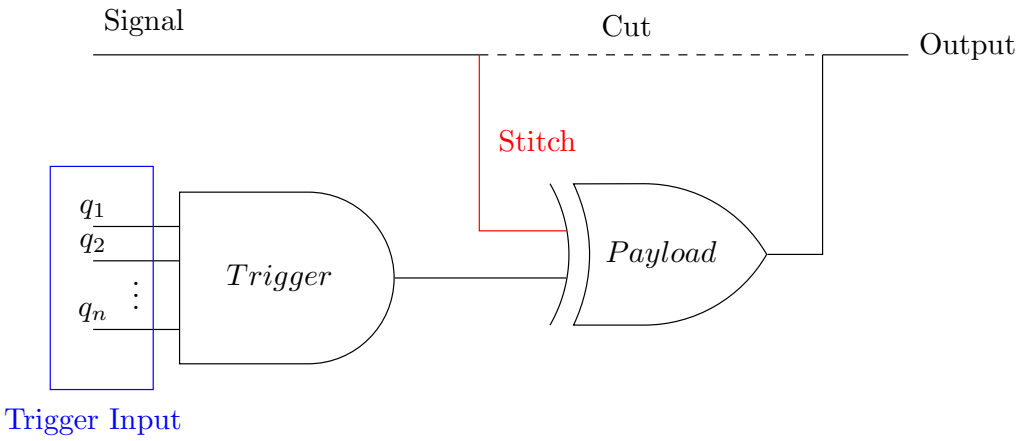


Figure 1.3. : Exemple d'un CTM

Le test fonctionnel [Wol+08] évite la destruction du circuit en testant son comportement à l'aide de vecteurs de tests. Un circuit comportant n entrées possède 2^n vecteurs de tests potentiels. En pratique, il est généralement impossible de tous les tester. Une partie de ces vecteurs est obtenue avec un algorithme générant un ensemble de vecteurs de test pour vérifier l'absence de défauts sur les signaux du circuit (*stuck-at 0/1*), appelé *Automated Test Pattern Generation (ATPG)*. Certains de ces vecteurs peuvent activer le CTM, modifiant les sorties attendues. Cette méthode peut être appliquée à tous les circuits après leur production, mais elle ne permet pas de détecter tous les CTMs, en particulier ceux ne modifiant pas les sorties.

Un circuit intégré a des propriétés physiques mesurables. Un changement mineur de *design*, tel qu'un CTM, perturbe ces propriétés. Une méthode appelée analyse de signal par canaux auxiliaires (*side channel signal analysis*) [Nar+10] consiste à comparer les propriétés d'un circuit non vérolé (*golden circuit*) avec celles des circuits produits. Les signaux comparés peuvent être électromagnétiques [Ngo+15], thermiques [CGM18], la consommation électrique [Agr+07], ainsi que le délai [JM08].

Des CTMs peuvent être ajoutés à des IPs tierces ou par un outil EDA malveillant. Certaines méthodes ont été proposées pour détecter des CTMs dans un *design* en utilisant des simulations. En générant des vecteurs de tests qui activent des signaux rares [Cha+09], principales entrées des *triggers*, il est possible de détecter des CTMs lors d'une simulation. Ces vecteurs peuvent ensuite être utilisés pour stimuler les CTMs et améliorer la détection post-fabrication. En complément de ces vecteurs de tests, les auteurs de [ZT11] proposent d'utiliser une méthode de couverture de code RTL. Des vecteurs de tests sont ajoutés pour contrôler des parties du code signalées comme suspectes. Enfin, les outils de méthodes formelles [RSK14b]

permettent de vérifier les fonctionnalités du circuit et de s'assurer qu'il n'est pas vérolé. Cependant, les méthodes formelles sont coûteuses en calcul, par conséquent, seules des parties limitées du circuit peuvent être examinées. De plus, seules les transitions d'états documentées sont vérifiées, car les transitions matériellement possibles sont exponentielles.

Récemment, de nouvelles méthodes utilisant l'Intelligence Artificielle (IA) ont été proposées dans le cadre de l'analyse de signaux par canaux auxiliaires [Gub+23]. Ces méthodes consistent à utiliser l'IA comme modèle prédictif des valeurs physiques pour remplacer un *golden circuit*, difficile à obtenir en pratique.

Les méthodes de détection tentent de dénicher un CTM, cependant, ne pas détecter un CTM ne garantit pas son absence. De plus, elles nécessitent une vérification pour chaque circuit. C'est pourquoi certains auteurs ont proposé des méthodes de prévention modifiant le *design* du circuit, permettant à chaque circuit de bénéficier de la sécurité apportée par la méthode. Ces méthodes sont appelées *Design for Hardware Trust (DfHT)* [TK10] et sont présentées ci-dessous.

1.1.3 Design for Hardware Trust

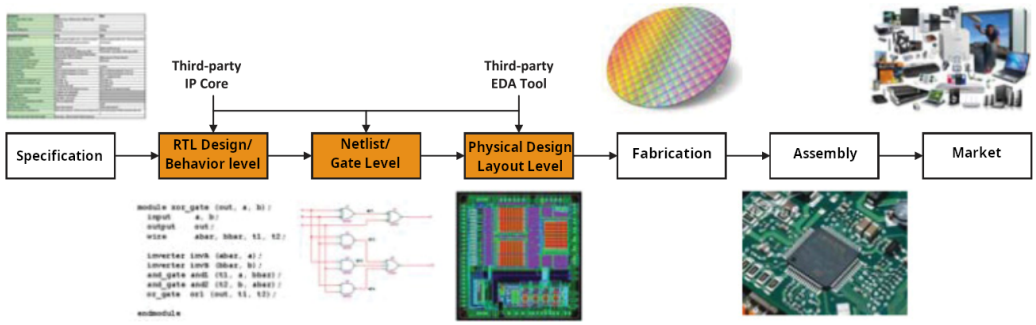


Figure 1.4. : Étape du flot de conception avec des méthodes de DfHT (orange)

Les méthodes de *DfHT* interviennent dans le flot de conception pour guider la conception du circuit, afin de prévenir l'insertion de CTM, faciliter leur détection ou fournir une authentification des circuits et des IPs. La Figure 1.4 surligne les étapes dans lesquels il y a des méthodes de *DfHT*.

Plusieurs méthodes ont été développées pour faciliter la détection des CTMs. L'inversion du voltage du circuit permet d'exacerber les anomalies de consommation, telles qu'un CTM, pour certains vecteurs d'entrée [BH09]. La réorganisation de la chaîne

de scan en régions permet d'effectuer des tests sur une partie isolée du circuit, facilitant ainsi la détection de CTMs induit par la limitation du bruit de consommation électrique du reste du circuit [STP10]. La réorganisation de l'arbre d'horloge du circuit permet de le segmenter en paires indépendantes de même consommation [Hos+17]. Cette réorganisation permet d'observer une modification de la consommation sur un segment en le comparant au segment appairé, supprimant ainsi la nécessité d'un *golden circuit*.

Lors de la fabrication, les circuits intégrés ont un taux de remplissage proche, mais rarement égal à 100%. En pratique, il reste souvent de l'espace pour de la logique, place que peut prendre un CTM. Combler cet espace empêcherait alors l'insertion de CTM. La méthode de *functional filler cell* [XFT14] consiste à remplir cet espace par des modules logiques intégrés au circuit. En conséquence, cette méthode modifie localement le circuit et impacte le délai.

Pour lutter contre la surproduction de circuits ou l'usage non contractuel d'IP, des méthodes d'authentification ont été créées. Elles verrouillent les circuits avec une clé numérique. Ce verrouillage peut se faire au niveau RTL avec un verouillage par automate fini *(Finite-State Machine locking, FSM locking)* [CB09 ; CB10]. Il utilise un automate à états fini qui demande une suite d'instructions précise comme clé numérique. Si cette suite d'instructions n'est pas exécutée correctement, le circuit se bloque dans un état d'erreur. Cette méthode ajoute du code RTL, ce qui va nécessairement augmenter la surface, la consommation et le délai du circuit. Un verrouillage peut également s'effectuer après synthèse, sur une *netlist*. Les méthodes de verrouillage logique *(logic locking)* [RKM08] consistent à ajouter des portes logiques dans le circuit, reliées à des bits de la clé numérique se trouvant dans une mémoire illisible [Raj+12a]. Ces portes logiques ajoutent des inconnues dans la fonction logique du circuit, empêchant ainsi sa compréhension et son bon fonctionnement. De plus, les auteurs de [Dup+14a] ont montré que ces méthodes peuvent également servir pour lutter contre l'insertion des CTMs en minimisant le nombre de signaux rares présents dans le circuit.

Les méthodes de prévention visent à éviter l'insertion d'un CTM, protégeant ainsi tous les circuits produits après la modification du *design*. Le *Logic Locking* répond à cette problématique en empêchant également la surproduction et en limitant l'ajout de CTM. Cependant, cette méthode modifie le circuit, ce qui peut nuire aux performances et augmenter la taille du circuit.

Cette thèse s'inscrit dans le cadre du projet ANR Multi-Objective Optimised Synthesis to Improve Cybersecurity (MOOSIC) en partenariat avec le CEA, le LIP6, le LIRMM ainsi qu'une entreprise privée Secure-IC. Chacun des partenaires apporte

une expertise pour la réalisation du projet. Le LIRMM proposent une expertise sur la sécurité matérielle, le CEA sur l'optimisation des méthodes en vue de la sécurité, et le LIP6 sur l'intégration de des méthodes dans un flot de conception ouvert. Secure-IC apporte un regard industriel au projet. Dans cette thèse effectuée au CEA et au LIP6, nous proposons d'explorer l'utilisation d'algorithmes d'optimisation pour maximiser la sécurité matérielle tout en limitant ces contre-coups dans un flot de conception ouvert. La section suivante détaille la problématique.

1.2 Problématique

L'augmentation de la production et de la complexité des circuits a engendré de nouvelles problématiques de sécurité. Parmi elles, l'insertion de CTMs représente un risque considérable pour les circuits utilisés dans des domaines tels que la santé, la défense ou le transport. De plus, la surproduction de circuits déstabilise l'économie du secteur et inflige d'importantes pertes financières. Plusieurs méthodes ont été proposées pour pallier ces problèmes. L'une d'entre elles s'avère prometteuse pour contrer ces quatres menaces : le *logic locking*. Cette méthode prévient l'insertion de CTMs en modifiant le circuit et limite la surproduction en utilisant une clé numérique. Toutefois, en modifiant le circuit, cette méthode impacte les performances, la consommation énergétique et les coûts de fabrication. Concilier la sécurité et les contre-coûts requiert une expertise élevée et laborieuse. Pour que des concepteurs non expert puissent sécurisé efficacement leurs circuits, il est nécessaire de proposer un outil automatique. Comme nous l'avons mentionné précédemment, les outils EDA peuvent manquer de fiabilité. Ainsi, la mise en œuvre d'outils automatiques dans des outils EDA open source constitue une réponse pertinente. Pour proposer de tels outils automatiques, nous devons utiliser des algorithmes capables d'optimiser la sécurité tout en limitant les impacts (surface / perfromance / puissance). Par conséquent, nous proposons d'utiliser des méthodes d'optimisation combinatoire et de recherche opérationnelle afin de trouver des compromis entre sécurité, délai et surface, répondant ainsi à ces exigences.

1.2.1 Contributions

Pour répondre à notre problématique, cette thèse propose :

- une extension de la mesure de sécurité présentée dans le *Strong Logic Locking* [Yas+16b] (Section 4.2)
- une modélisation mathématique du problème de sécurité (Section 4.4)
- des algorithmes exacts avec différentes approches pour le résoudre (Section 4.5)
- une modélisation temporelle du problème, sans prendre en compte la sécurité (Section 5.2)
- l'ajout d'une contrainte temporelle dans le modèle de sécurité (Section 5.3.1)
- un modèle multi-objectif sécurité / temporelle / surface (Section 5.3.2)
- une extension de sécurité pour un flot de conception ouvert (Section 5.4)

1.2.2 Organisation du manuscrit

Cette thèse s'inscrit dans deux domaines : électronique et optimisation. Le chapitre 2 introduit des notions utilisées pour permettre une meilleure compréhension des concepts utilisés. Le chapitre 3 détaille différentes approches du *logic locking*, ainsi que les outils EDA liés à la sécurité. Le chapitre 4 enchaîne sur la modélisation et la résolution du *Strong Logic Locking*. Le chapitre 5 propose une analyse temporelle de l'insertion d'un verouillage, ainsi que l'intégration de nos algorithmes dans une extension de sécurité pour le flot de conception.

Notions et définitions

2

Sommaire

Cette thèse a pour but de proposer des algorithmes d'optimisation pour la conception automatique de circuits intégrés. Elle s'inscrit dans deux domaines : la conception en électronique et la recherche opérationnelle. Dans ce chapitre, nous allons présenter les notions mathématiques et électroniques nécessaires pour comprendre la thèse. Dans la section mathématique, nous présenterons la théorie des graphes, la recherche opérationnelle et la complexité algorithmique. Dans la section électronique, nous présenterons les notions élémentaires pour les circuits combinatoires, ainsi que la notion de délai et de surface. Ces notions seront utilisées tout au long du manuscrit.

2.1 Notions mathématiques

Dans cette section, nous présentons et définissons les notions mathématiques abordées dans la thèse. Nous commençons par définir les graphes qui sont utilisés pour représenter des circuits. Nous poursuivons avec la présentation de la Recherche Opérationnelle (RO), et terminons par une brève explication sur la complexité algorithmique.

2.1.1 Définitions de graphe

Nous définissons les notions élémentaires des graphes pour les réutiliser dans la thèse. Ces définitions sont inspirées de [Ber85].

$G1$:

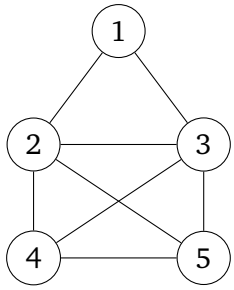


Figure 2.1. : Exemple d'un graphe non orienté

$G2$:

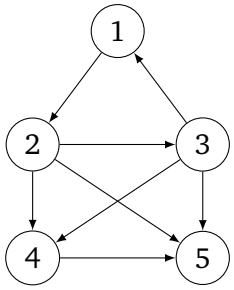


Figure 2.2. : Exemple d'un graphe orienté

Définition 2.1.1 (*Sommet*) : *Les sommets sont les composants atomiques de la théorie des graphes. Ils sont parfois appelés nœuds selon le contexte, mais représentent le même objet. L'ensemble des sommets est généralement noté V pour vertices en anglais.*
Exemple des graphes de la Figure 2.1 et Figure 2.2 : l'ensemble des sommets de $G1$ et $G2$ est $V = \{1, 2, 3, 4, 5\}$.

Définition 2.1.2 (*Arête*) : *Les arêtes sont des paires de sommets (un ensemble à deux élements). L'ensemble des arêtes est généralement noté E pour edges en anglais. Elles sont non orientées, i.e. $\forall e = \{v_1, v_2\} \in E,\ \{v_1, v_2\} = \{v_2, v_1\}$.*
Exemple de la Figure 2.1 : l'ensemble des arêtes de $G1$ est
$E = \{\{1,2\}, \{1,3\}, \{2,3\}, \{2,4\}, \{2,5\}\ \{3,4\}, \{3,5\}, \{4,5\}\}$.

Définition 2.1.3 (*Arc*) : *Les arcs sont des couples de sommets (un vecteur à deux élements). L'ensemble des arcs est généralement noté A. Ils sont orientés, i.e. $\forall a = (v_1, v_2) \in A,\ (v_1, v_2) \neq (v_2, v_1)$. Dans un arc, il y a un sommet de départ et d'arrivée.*
Exemple de la Figure 2.2 : l'ensemble des arcs de $G2$ est
$A = \{(1,2), (2,3), (2,4), (2,5)\ (3,1), (3,4), (3,5), (4,5)\}$.

Définition 2.1.4 (*Graphe non orienté*) : *Un graphe non orienté (undirected graph) $G = (V, E)$ est un couple formé d'un ensemble sommets V et d'un ensemble d'arêtes E.*
Le graphe $G1$ de la Figure 2.1 est non orienté.

Définition 2.1.5 (*Graphe orienté*) : *Un graphe orienté (directed graph) $G = (V, A)$ est un couple formé d'un ensemble sommets V et d'un ensemble d'arcs A.*
Le graphe $G2$ de la Figure 2.2 est orienté.

Définition 2.1.6 (*Sommets adjacents*) : *Soit un graphe $G = (V, E)$ non orienté et un graphe $G' = (V, A)$ orienté.*

— ***Graphe non orienté :*** *Un sommet u est adjacent à un sommet v dans G s'il existe une arête $\{u, v\} \in E$. Par exemple, dans la Figure 2.1, 2 est adjacent à 1, mais 5 n'est pas adjacent à 1.*

— ***Graphe orienté :*** *Un sommet u est adjacent à un sommet v dans G' s'il existe un arc $(u, v) \in A$ ou $(v, u) \in A$. Par exemple, dans la Figure 2.2, 2 est adjacent à 1, mais 1 n'est pas adjacent à 5.*

Définition 2.1.7 (*Degré*) : *Soit un graphe $G = (V, E)$ non orienté et un graphe $G' = (V, A)$ orienté.*

— ***Graphe non orienté :*** *Le degré d'un sommet v, noté $\delta(v)$, est le nombre d'arêtes contenant v. Par exemple, dans la Figure 2.1, $\delta(1) = 2$ et $\delta(2) = 4$.*

— ***Graphe orienté :***

  — *Le degré entrant d'un sommet v, noté $\delta^-(v)$, est le nombre d'arcs dirigés vers v.*

  — *Le degré sortant d'un sommet v, noté $\delta^+(v)$, est le nombre d'arcs sortant de v.*

  — *Le degré d'un sommet v, noté $\delta(v)$, est la somme du degré entrant et sortant de v, soit $\delta(v) = \delta^-(v) + \delta^+(v)$. Par exemple, dans la Figure 2.2, $\delta(2) = \delta^-(2) + \delta^+(2) = 1 + 3 = 4$.*

Définition 2.1.8 (*Graphe complet*) : *Un graphe complet est défini par $G = (V, V^2)$. Toutes paires de sommets $(u, v) \in V^2$, u est adjacent à v, c'est-à-dire qu'il existe une arête (ou un arc) entre u et v. Pour un ensemble de sommets donné, il existe un unique graphe complet.*
La Figure 2.3 est le graphe complet à 5 sommets.

Définition 2.1.9 (*Graphe vide*) : *Un graphe vide est défini par $G = (V, \emptyset)$, c'est-à-dire un graphe sans arête ni arcs.*

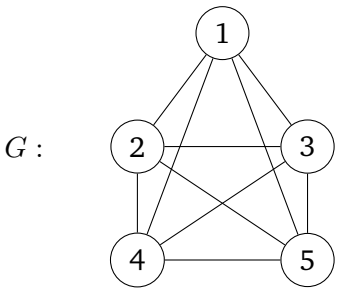


Figure 2.3. : Graphe complet à 5 sommets

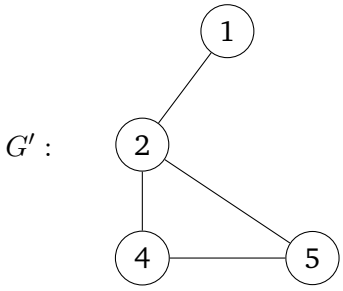


Figure 2.4. : Sous-graphe induit du graphe G1 de la Figure 2.1 par $V' = \{1, 2, 4, 5\}$

Définition 2.1.10 (*Sous-graphe induit*) : *Soit un graphe $G = (V, E)$. Un sous-graphe induit $G' = (V', E')$ de G est tel que $V' \subseteq V$ et $E' = \{\{u, v\} \in E \mid u \in V' \text{ et } v \in V'\}$. Cela signifie que G' est composé d'un sous-ensemble de sommets, et que les arêtes (ou arcs) de G' sont des arêtes qui existent dans G et que toutes les arêtes de G entre les sommets du sous-ensemble sont dans G'. Pour un graphe G et un sous-ensemble de sommets donné, il existe un unique sous-graphe induit.*
La Figure 2.4 est le sous-graphe induit de la Figure 2.1 avec $V' = \{1, 2, 3, 4\}$.

Définition 2.1.11 (*Chaîne et Chemin*) : *Soit un graphe non orienté $G = (V, E)$ et un graphe orienté $G' = (V', A)$.*

- *Chaîne (dans un graphe non orienté) : Une chaîne dans G est une suite consécutive de sommets reliés par des arêtes. Formellement, une séquence $(v_1, v_2, \ldots, v_k)$ est une chaîne si pour tout i avec $1 \leq i < k$, l'arête $\{v_i, v_{i+1}\}$ appartient à E.*
  Dans la Figure 2.5, $(1, 2, 3)$ est une chaîne, $(1, 3, 4, 5)$ est aussi une chaîne, mais $(1, 5, 4)$ n'est pas une chaîne car $\{1, 5\} \notin E$.
- *Chemin (dans un graphe orienté) : Un chemin dans G' est une suite consécutive de sommets reliés par des arcs. Formellement, une séquence $(v_1, v_2, \ldots, v_k)$ est un chemin si pour tout i avec $1 \leq i < k$, l'arc (v_i, v_{i+1}) appartient à A.*
  Dans la Figure 2.6, $(1, 2, 3)$ est un chemin, $(1, 2, 3, 4, 5)$ est aussi un chemin, mais $(1, 3, 4, 5)$ n'est pas un chemin car $(1, 3) \notin A$.

Notons que, par définition, une sous-chaîne (resp. un sous-chemin) est une chaîne (resp. un chemin).

$G1$:

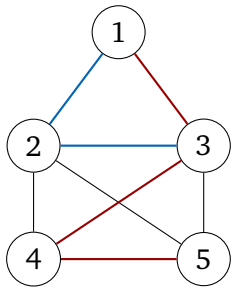


Figure 2.5. : Exemple de chaînes $(1, 2, 3)$ en bleu et $(1, 3, 4, 5)$ en rouge

$G2$:

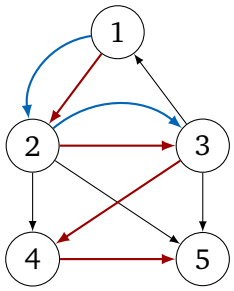


Figure 2.6. : Exemple de chemins $(1, 2, 3)$ en bleu et $(1, 2, 3, 4, 5)$ en rouge

Définition 2.1.12 (*Graphe connexe*) :

Graphe non orienté : *Un graphe non orienté est connexe si, pour toutes paires de sommets (u, v), il existe une chaîne entre u et v.*

Graphe orienté : *Un graphe orienté est dit :*

— *Faiblement connexe si, pour toutes paires de sommets (u, v), en ignorant le sens des arcs, il existe une chaîne entre u et v.*

— *Fortement connexe si, pour toutes paires de sommets (u, v), il existe un chemin de u vers v.*

Définition 2.1.13 (*Clique*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$). Une clique $c \subseteq V$ est un sous-ensemble de sommets qui induit un sous-graphe complet. Donc, $\forall v_1, v_2 \in c, \{v_1, v_2\} \in E$ (resp. $(v_1, v_2) \in A$ **et** $(v_2, v_1) \in A$). Autrement dit, $c^2 \subseteq E$ (resp. $c^2 \subseteq A$).*
Par exemple dans la Figure 2.1, l'ensemble $\{2, 3, 4\}$ est est une clique car chaque sommet de cette ensemble est relié aux autres.

Définition 2.1.14 (*Clique maximum*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$), soit $C(G)$ l'ensemble des cliques de G. Une clique maximum $\overline{c} \in C(G)$ est une des cliques de plus grande taille (au sens de la cardinalité) de G, i.e. $|\overline{c}| = \max_{c \in C(G)} |c|$*

Définition 2.1.15 (*Clique maximale*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$), soit $C(G)$ l'ensemble des cliques de G. Une clique maximale $c \in C(G)$ est une clique qui n'est pas un sous-ensemble d'une autre clique. Elle est dite maximale au sens de l'inclusion. Donc, $c \in C(G)$ est maximale si et seulement si $\forall c' \in C(G) \backslash \{c\},\ c \nsubseteq c'$. Autrement dit, $c \in C(G)$ est maximale si $\forall v \in V \backslash c,\ \exists v' \in c, (v, v') \notin E$ (resp. $(v, v') \notin A$ **ou** $(v', v) \notin A$).*
Dans la Figure 2.1, la clique $\{1, 2, 3\}$ est une clique maximale. Aucune autre clique contient ces trois éléments.

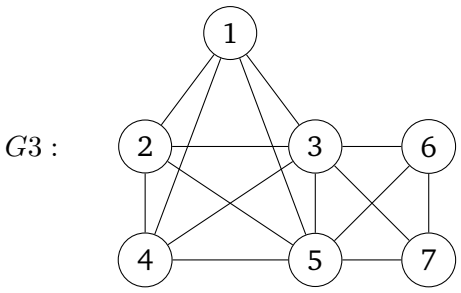


Figure 2.7. : Graphe non orienté

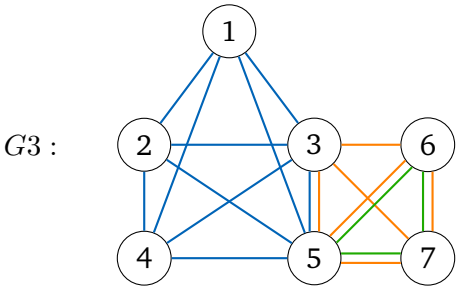


Figure 2.8. : Exemple dans $G3$ d'une clique $(5, 6, 7)$ en vert, d'une clique maximale $(3, 5, 6, 7)$ en orange et d'une clique maximum $(1, 2, 3, 4, 5)$ en bleu

Example 2.1.1 (*Clique maximum ≠ maximale*) : *Calculons l'ensemble des cliques de $G3$ de la Figure 2.8 :*

taille	*cliques*
1	*{1}, {2}, {3}, {4}, {5}, {6}, {7}*
2	*{1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 5}, {5, 6}, {5, 7}, {6, 7}*
3	*{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {1, 3, 5}, {1, 4, 5}, {2, 3, 4}, {2, 3, 5}, {2, 4, 5}, {3, 4, 5}, {3, 5, 6}, {3, 5, 7}, {3, 6, 7}, {5, 6, 7}*
4	*{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}, {3, 5, 6, 7}*
5	*{1, 2, 3, 4, 5}*

Nous pouvons maintenant calculer l'ensemble des cliques maximales, i.e. ceux qui ne sont pas sous-ensembles d'une autre clique.

taille	*cliques maximales*
1	$\emptyset$
2	$\emptyset$
3	$\emptyset$
4	*{3, 5, 6, 7}*
5	*{1, 2, 3, 4, 5}*

La clique de $G3$ comportant le plus d'élément est $\{1, 2, 3, 4, 5\}$. Elle est la plus grande clique , elle est donc une clique maximum. La clique {3, 5, 6, 7} est également une clique maximale, mais n'est pas maximum. Notons qu'une clique maximum est nécessairement maximale, l'inverse n'est pas vrai.

Définition 2.1.16 (*Stable*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$). Un stable $s \subseteq V$ est un sous-ensemble de sommets qui induit un sous-graphe vide. Donc, $\forall v_1, v_2 \in s, \{v_1, v_2\} \notin E$ (resp. $(v_1, v_2) \notin A$ **et** $(v_2, v_1) \notin A$). Autrement dit, $c^2 \cap E = \emptyset$ (resp. $c^2 \cap A = \emptyset$).*
Dans la Figure 2.1, l'ensemble $\{1, 2\}$ forment un stable.

Définition 2.1.17 (*Stable maximum*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$), soit $S(G)$ l'ensemble des stables de G. Un stable maximum $\overline{s} \in S(G)$ est un stable de plus grande taille de G, i.e. $|\overline{s}| = \max_{s \in S(G)} |s|$*

Définition 2.1.18 (*Stable maximale*) : *Soit un graphe non orienté $G = (V, E)$ (resp. orienté $G = (V, A)$), soit $S(G)$ l'ensemble des stables de G. Un stable maximale $s \in S(G)$ est un stable qui n'est pas un sous-ensemble d'un autre stable. Il est dit maximale au sens de l'inclusion. Donc, $s \in S(G)$ est maximale si et seulement si $\forall s' \in S(G) \backslash \{s\},\ s \nsubseteq s'$. Autrement dit, $s \in S(G)$ est maximale si $\forall v \in V \backslash s,\ \exists v' \in s, (v, v') \in E$ (resp. $(v, v') \in A$ **ou** $(v', v) \in A$).*

2.1.2 Introduction à la Recherche Opérationnelle

La recherche opérationnelle utilise les mathématiques pour optimiser les décisions dans des systèmes complexes. Son application va de la planification industrielle à la gestion automatique d'emplois du temps en passant par la logistique et l'*EDA*. La recherche opérationnelle permet une prise de décision automatisée en s'appuyant sur des outils mathématiques.

Cependant, ces outils nécessitent un travail de modélisation du problème concret à un problème mathématique. Une fois le problème modélisé, nous pouvons le résoudre de manière optimale, trouver la meilleure solution, ou de manière approchée, trouver une bonne solution avec ou sans garantie. Ce choix détermine les algorithmes utilisés pour la résolution, et leur rapidité d'exécution.

En pratique, trouver une solution optimale nécessite plus de temps de calcul, car une partie est dédiée à prouver qu'aucune meilleure solution n'existe. Cependant, certains problèmes sont simples et peuvent être résolus optimalement avec des algorithmes rapides. Des algorithmes génériques permettent de résoudre tous les modèles mathématiques linéaires qui peuvent représenter de nombreux problèmes. Ces algorithmes sont implémentés dans des outils appelés solveurs : Gurobi, CPLEX, Z3, Choco, ... Les solveurs peuvent résoudre tout les modèles mathématiques linéaires mais ces algorithmes ont un temps d'éxécution exponentiel. La résolution peut aussi s'effectuer avec des algorithmes spécifiques pour le problème modélisé. Ces algorithmes sont dits dédiés au problème et reposent généralement sur des principes algorithmiques comme le *branch and bound* ou la programmation dynamique. Les algorithmes dédiés s'appuient sur une particularité du problème qui permet parfois une résolution en temps polynomial. Un exemple connu est le problème du plus court chemin entre deux points dans un graphe.

Trouver une solution optimale peut parfois être coûteux en calcul. Il peut être intéressant de trouver rapidement des solutions non-optimales grâce à des algorithmes spécifiques : algorithme glouton, recherche locale, recuit simulé, ... Ces algorithmes

sont appelés des heuristiques et reposent sur des principes simples permettant de s'adapter à chaque problème et de calculer des solutions rapidement.

Dans certains cas, les problèmes étudiés possèdent plusieurs objectifs. L'optimisation multi-objectifs est un champ d'étude regroupant des outils mathématiques spécifiques pour résoudre ce cas. Comme il existe rarement une solution optimale qui répond à tous les objectifs, les outils retournent un ensemble de solutions qui ne sont pas meilleures les unes par rapport aux autres. Ces solutions sont appelées solutions non dominées et forment un ensemble appelé front de Pareto optimal.

2.1.3 Notions pour la théorie de la complexité

L'étude de la complexité des problèmes est un champ d'étude important en informatique théorique. La complexité d'un problème se définit par la complexité algorithmique du meilleur algorithme connu pour résoudre ce problème [GJ79]. Par exemple, le problème du plus court chemin dans un graphe peut être résolu par l'algorithme de Dijkstra [Dij59], qui a une complexité algorithmique en $O\left((n+m)\log_2(n)\right)$, avec n le nombre de sommets et m le nombre d'arcs. Cette complexité algorithmique est dite polynomiale, car elle est bornée par une fonction polynomiale, par exemple : $(n+m)^2$. Lorsqu'il existe un algorithme polynomial pour résoudre un problème, il appartient à la classe de complexité P.

Il existe une classe de complexité appelée NP qui est un ensemble de problèmes qui possèdent un algorithme de vérification permettant de déterminer en un temps polynomial si une solution donnée est valide. S'il existe un algorithme qui trouve une solution en un temps polynomial, il existe nécessairement un algorithme polynomial pour la vérification. On en conclut que $P \subset NP$. Certains problèmes dans NP peuvent se transformer en d'autres par un procédé appelé une **réduction**. Tous les problèmes de NP peuvent se réduire au problème de satisfiabilité d'une fonction logique, appelé problème SAT, en un temps polynomial [Coo71]. Le problème SAT appartient alors à la classe $NP-complet$ car sa résolution permet de résoudre tous les problèmes NP. Plusieurs problèmes ont été démontrés comme appartenant à la classe $NP-complets$ [Kar75] et possèdent tous cette propriété. Néanmoins, nous ignorons aujourd'hui si ces problèmes possèdent un algorithme de résolution en temps polynomial. Si tel est le cas, alors nous pourrions résoudre tous les problèmes NP en temps polynomial. Certains scientifiques considèrent qu'il n'existe pas d'algorithme polynomial pour résoudre les problèmes $NP-complets$, mais que la preuve de leur inexistence est difficile à construire. Il est alors fréquent de voir des preuves de complexité commencer par "Sous l'hypothèse que $P \neq NP$, ... ".

Lorsqu'un nouveau problème d'optimisation est étudié, il est fréquent d'essayer de l'attribuer à une classe de complexité. Certains problèmes $NP - complets$ peuvent se réduire à ce nouveau problème, ce qui le fait intégrer dans la classe de complexité $NP\text{-}DUR$. Un problème dans cette classe de complexité indique que nous ne connaissons pas d'algorithme polynomial permettant de résoudre le problème, et qu'en trouver un relève d'une avancée scientifique très attendue. Il est raisonnable de considérer qu'un problème $NP\text{-}DUR$ ne peut pas être résolu en un temps polynomial. Il est donc nécessaire de proposer un algorithme exponentiel qui fonctionne en pratique sur des tailles d'instances suffisamment petites, ou de proposer des résolutions approchées.

Il existe des classes de problèmes d'optimisation traitant de l'approximabilité avec garantie. Ces classes permettent de trier les problèmes en fonction de leur difficulté d'approximation par rapport à leur valeur optimale. L'une d'entre elles est la classe APX (abréviation de « approximable »). Si un problème est dans la classe APX, alors il existe un algorithme avec un ratio d'approximation constant r, permettant de trouver une solution approchée $\tilde{z}$ telle que $\frac{1}{r}z^* \leq \tilde{z} \leq z^*$, avec z^* la valeur optimale du problème d'optimisation (dans le cas d'une maximisation).

Prouver qu'un problème est $NON\text{-}APX$ signifie qu'il n'existe pas d'algorithme d'approximation avec un ratio constant qui s'exécute en un temps polynomial. Un exemple est le problème de la clique maximum, qui consiste à trouver la taille de la plus grande clique dans un graphe [Zuc06].

2.2 Notions de l'électronique

Dans cette thèse, nous travaillons plus spécifiquement sur les circuits combinatoires. Ces circuits sont composés d'entrées et de sorties qui peuvent être des registres ou des connecteurs extérieurs. Entre ces entrées et sorties, le circuit encode une fonction logique booléenne avec des portes logiques et des signaux. Chaque porte logique effectue elle-même une fonction logique. Dans notre cas, un circuit combinatoire sera représenté par des interconnexions de portes (*netlist* en anglais), sous la forme d'un graphe, avec les sommets correspondant aux portes logiques et les arêtes représentant les signaux électroniques. Dans un circuit combinatoire, les signaux sont orientés de la sortie des portes vers l'entrée des portes logiques suivantes. Cette orientation est induite, et n'est pas représentée graphiquement.

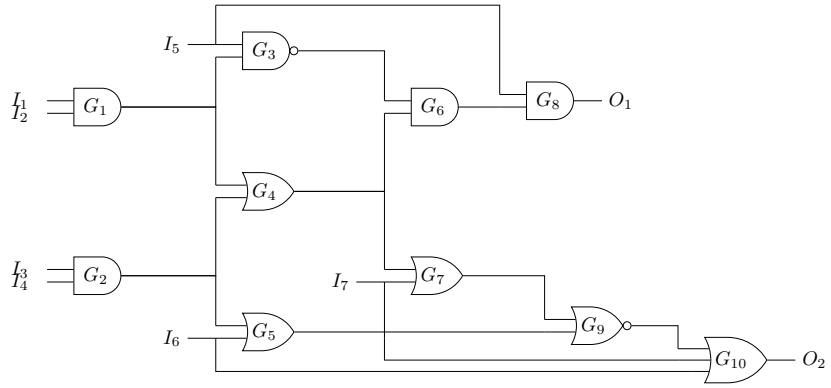


Figure 2.9. : Exemple de circuit sous forme de *netlist*

La Figure 2.9 est un exemple de circuit combinatoire représenté sous forme de *netlist*. Nous pouvons distinguer les portes logiques avec une forme représentant une fonction logique. Les portes G_1, G_2, G_6, G_8 sont des portes AND. Elles effectuent un ET logique avec leurs signaux d'entrées. Les portes $G_4, G_5, G_7, G_9, G_{10}$ sont des portes OR. La porte G_3 est une porte $NAND$. Enfin, la porte G_9 est une porte NOR. Les tables de vérité de ces portes sont décrites dans le Tableau 2.1.

A	B	$A \wedge B$ (AND)	$\neg(A \wedge B)$ (NAND)	$A \vee B$ (OR)	$\neg(A \vee B)$ (NOR)
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

Tableau 2.1. : Tableau des valeurs logiques pour les opérations AND, NAND, OR et NOR

La distance de Hamming est très utilisée en sécurité matérielle. Elle permet de mesurer la différence entre deux vecteurs booléens, qui peuvent être des vecteurs d'entrées ou de sorties.

Définition 2.2.1 (*Distance de Hamming*) : *La distance de Hamming est une mesure de distance entre deux vecteurs v, v' de taille n. Elle se définit par la somme des différences entre chaque champs des vecteurs :*

$$DH(v, v') = \sum_{i=1}^{n} |v_i - v'_i|$$

Example 2.2.1 (*Exemple de distance de Hamming*) : *Dans le circuit Figure 2.9, il y a 7 entrées dans le circuit. Un vecteur d'entrée du circuit possède alors 7 valeurs. Prenons*

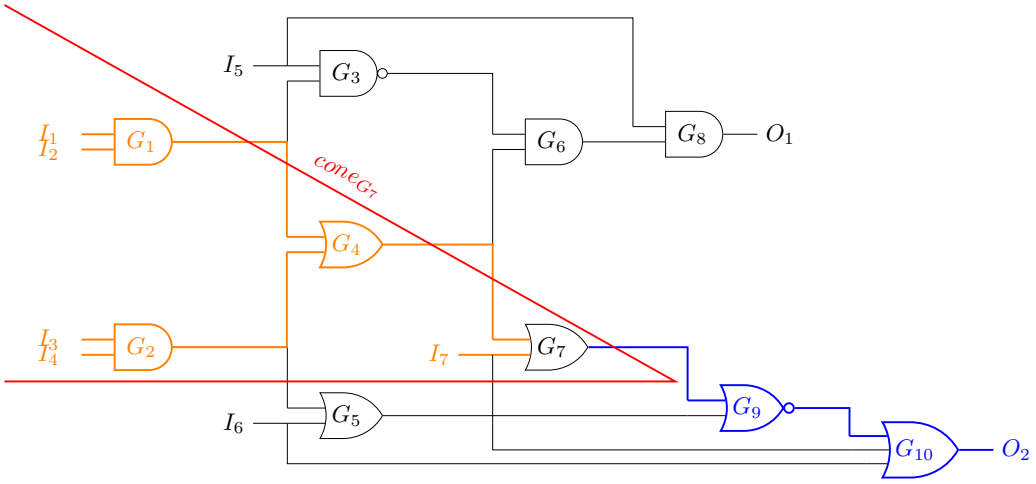


Figure 2.10. : Circuit avec le *fan-in tree* (orange), le *fan-out tree* (bleu) et le cone logique (rouge) de G_7

deux vecteurs d'entrées, $I = (0,0,0,0,0,0,0)$ *et* $I' = (1,0,1,1,0,1,0)$. *La distance de Hamming vaut :* $\sum_{j=1}^{7} \left| I_j - I'_j \right|$, *dans notre cas, la distance de Hamming entre* I *et* I' *vaut* 4.

L'ensemble des entrées, des signaux et des portes logiques ayant une influence logique sur une porte logique g est appelé le ***fan-in tree*** de g. L'union d'une porte g et de son *fan-in tree* est appelée **cône logique**. L'ensemble des sorties, des signaux et des portes logiques sur lesquels une porte g à une influence logique est appelée ***fan-out tree***. La Figure 2.10 représente un circuit combinatoire avec le *fan-in tree* de G_7 en orange, le cône logique de G_7 en rouge et le *fan-out tree* de G_7 en bleu.

Dans un circuit combinatoire, le **délai de propagation** correspond au temps nécessaire pour mettre à jour la sortie d'une porte logique après une modification d'un signal d'entrée du circuit. Le délai de propagation d'un porte g correspond alors au temps minimum pour mettre à jour tous les signaux du cône logique de g pour s'assurer que le calcul soit effectué. Le temps de propagation d'un circuit est le maximum des délais de propagation des sorties du circuits. Ce temps détermine la fréquence d'utilisation du circuit, et donc ses performances. Chaque porte logique a un temps de propagation qui dépend de la technologie de gravure. De plus, chaque signal a également un temps de propagation. Ce délai dans les signaux dépend essentiellement de l'étape de placement-routage. Plus une technologie de gravure est fine, plus le rapport entre les temps de propagation des fils et le temps de propagation des portes est important. Dans notre cas, nous utilisons des technologies de gravure matures (≥ 100nm) où le délai de propagation des fils est négligeable, ce qui n'est pas le cas des technologies avancées. Nous calculons alors le délai de propagation

pour une porte g comme : $d(g) = T_g + \max_{g' \in \delta^-(g)} d(g')$ avec T_g le temps de propagation d'une porte.

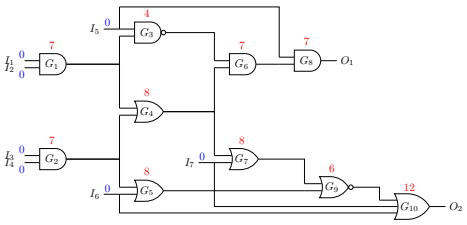
Example 2.2.2 (*Calcul du temps de propagation*) : *La Figure 2.11 calcule le temps de propagation du circuit de la Figure 2.9. Les délais sont ici en nanosecondes.*

On initialise le temps de propagation des entrées à 0, on calcul le délai pour chaque porte :

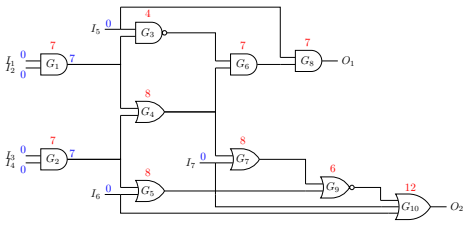
$$d(G_1) = d(G_2) = 7 + \max\{0, 0\} = 7 \tag{2.1}$$
$$d(G_3) = 4 + \max\{0, 7\} = 11 \tag{2.2}$$
$$d(G_4) = 8 + \max\{7, 7\} = 15 \tag{2.3}$$
$$d(G_5) = 8 + \max\{0, 7\} = 15 \tag{2.4}$$
$$d(G_6) = 7 + \max\{11, 15\} = 22 \tag{2.5}$$
$$d(G_7) = 8 + \max\{15, 0\} = 23 \tag{2.6}$$
$$d(G_8) = 7 + \max\{0, 22\} = 29 \tag{2.7}$$
$$d(G_9) = 6 + \max\{23, 15\} = 29 \tag{2.8}$$
$$d(G_{10}) = 12 + \max\{29, 0, 0\} = 41 \tag{2.9}$$

Le délai maximum : $\max_{g \in V} d(g) = 41$ *correspond au temps nécessaire pour que les signaux électriques soient mis à jour dans le circuit pour que le calcul soit cohérent. Nous obtenons alors un chemin critique composé de* $\{I_1, I_2, I_3, I_4, G_1, G_2, G_4, G_7, G_9, G_{10}\}$.

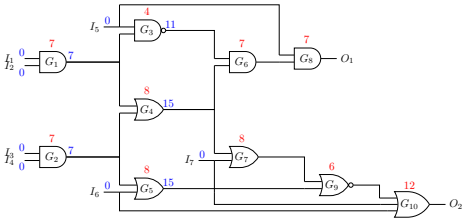
Dans cette thèse, nous manipulons des circuits combinatoires sous forme de *net-list*. Ces circuits sont amenés à être fabriqués. Leur coût de fabrication dépend de leur taille, mesurée en surface. La taille est déterminée par la disposition des portes puisque les fils passent au-dessus des portes. Les outils de placement-routage cherchent à limiter l'espace utilisé par un circuit. Lors de l'étape de la synthèse, les informations sur la taille des portes sont connues. Nous utilisons alors la somme des tailles des portes logiques pour estimer la surface d'un circuit.



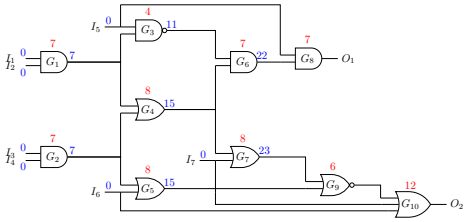
(a) Temps de propagation des portes en rouge et intialisation des temps de propagation des entrées en bleu



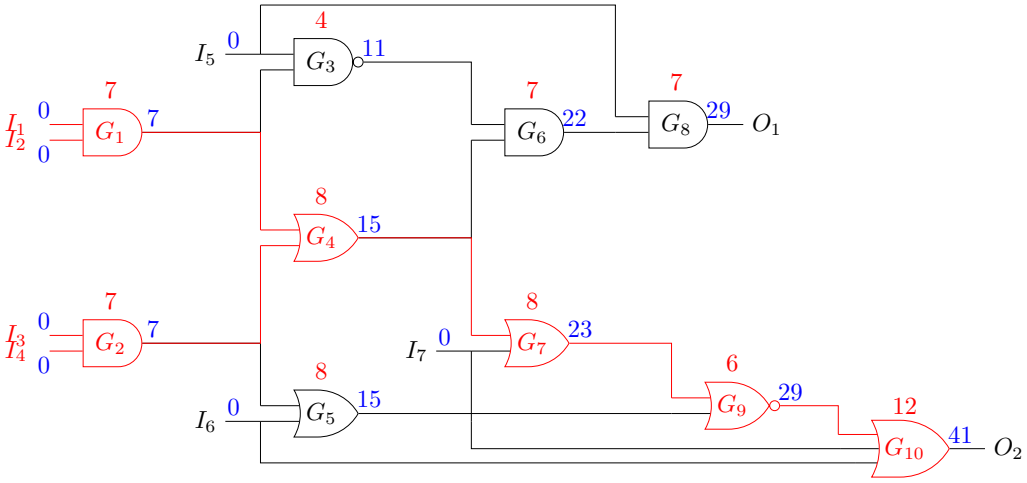
(b) Calcul des temps de propagation des portes G_1 et G_2



(c) Calcul des temps de propagation des portes G_3, G_4 et G_5



(d) Calcul des temps de propagation des portes G_6 et G_7



(e) Calcul des temps de propagation des portes G_8, G_9 et G_{10} avec les chemins critiques en rouges

Figure 2.11. : Calcul du délai de propagation du circuit en bleu avec le délai de propagation de chaque porte

Example 2.2.3 (*Exemple de calcul de surface du circuit Figure 2.9 pour une technologie 350nm*) : *Dans le circuit de la Figure 2.9, nous avons :* 4 *portes* AND, 1 *porte* $NAND$, 3 *portes* OR, 1 *porte* NOR *et* 1 *porte* $OR3$. *On prend leur taille en* nm^2 *et on les additionne (les tailles données servent d'exemple) :* $4 * 180 + 1 * 140 + 3 * 140 + 1 * 140 + 1 * 210 = 1630nm^2$. *On estime alors la taille du circuit à* $1630nm^2$.

2.3 Conclusion

Nous avons définit les notions élémentaires de la théorie des graphes, de la recherche opérationnelle, de la complexité et de l'électronique pour la suite de la thèse. Elles aident à la compréhension du sujet, de la littérature et des travaux présentés. Nous pouvons maintenant présenter un état de l'art sur le *logic locking* dans le chapitre suivant.

3 État de l'art

Sommaire

3.1 Introduction

Dans l'introduction, nous avons constaté qu'une méthode efficace utilisée dans le DfHT pour lutter contre le piratage est le *logic locking*. Une abondante littérature existe sur le sujet, proposant différentes méthodes. Dans ce chapitre, nous présentons diverses approches du *logic locking*, leur fonctionnement et leurs objectifs. Ces méthodes sont exposées dans un ordre historique, en commençant par les méthodes *pré-SAT*, suivies par la présentation de l'attaque SAT visant à supprimer le verouillage, pour ensuite aborder les méthodes *post-SAT*.

3.2 Logic locking

Le *logic locking*, parfois appelé *logic obfuscation* ou *logic encryption* [DF19], est une méthode initialement utilisée pour lutter contre la surproduction et le vol d'IP, qui peut également être utilisée contre l'insertion de chevaux de Troie [Dup+14b]. Cette méthode consiste à verrouiller logiquement le circuit, avec une **clé numérique**. Sans la bonne clé, le circuit est dysfonctionnel, avec la bonne clé, le circuit est logiquement identique à une version non verrouillée. Cette clé numérique est reliée à des portes logiques additionnelles, appelées **portes clés**. Chaque porte clé est reliée à un bit de

la clé numérique. La clé numérique choisis par le constructeur pour déverrouiller le circuit est appelé **clé secrète**.

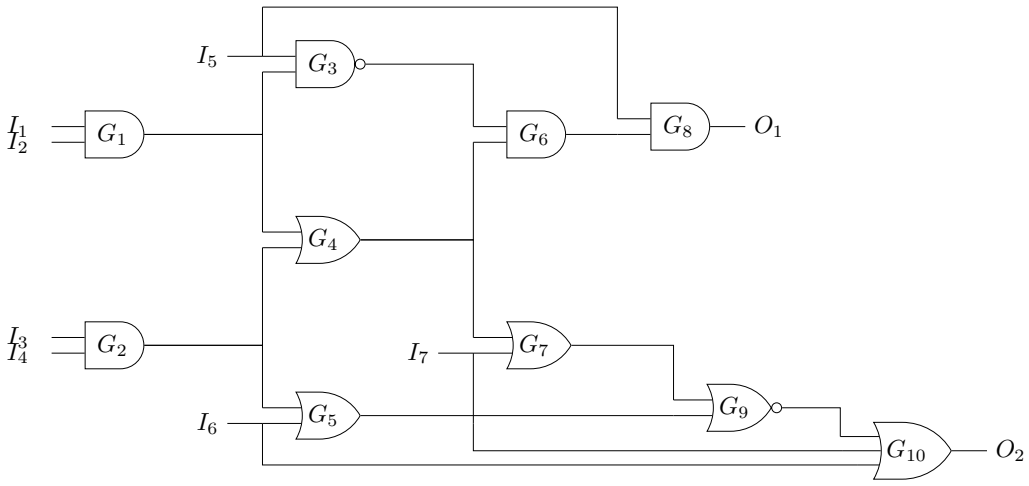
Dans un circuit verrouillé, on considère que la clé numérique est illisible [Raj+12a]. Pour ce faire, chaque clé est composée d'un bit de clé utilisateur U_{key} et d'une fonction physique inclonable (*Physical Unclonable Function : PUF*). Un *PUF* [SD07] est un circuit qui, pour une entrée $P_{challenge}$, répond par une sortie $P_{response}$. La réponse d'un PUF ne peut pas être prédite car elle dépend des délais physiques du circuit. Elle est unique pour chaque circuit fabriqué. Le fabricant du circuit doit alors transmettre les tables de correspondance entre $P_{challenge}$ et $P_{response}$ pour chaque circuit. Le concepteur peut alors déterminer le U_{key} unique pour chaque circuit, et le transmettre à l'utilisateur. Ainsi, même en connaissant la *netlist* du circuit, il est impossible de déterminer la valeur de sortie d'un *PUF*.

Example 3.2.1 (*Exemple de logic locking illustré dans la Figure 3.1*) : *Le circuit de la Figure 3.1a est verrouillé en ajoutant une clé numérique* $K = (k_1, \ldots, k_7)$ *et des portes clés sur certains signaux. Chaque bit de clé* k_i *est construit avec un couple* $(P_{challenge-i}, U_{key-i})$. $P_{challenge-i}$ *est utilisé dans un* PUF*, qui fournit* $P_{response-i}$*, une valeur inconnue pour l'utilisateur.* k_i *est le résultat de* $XOR(P_{response-i}, U_{key-i})$ *et est relié à une porte clé* K_i *dans le circuit. On obtient ainsi le circuit verrouillé avec l'illustration du bit de clé* k_7 *dans la Figure 3.1b. Les signaux rouges représentent des signaux inconnus, sauf pour le concepteur.*

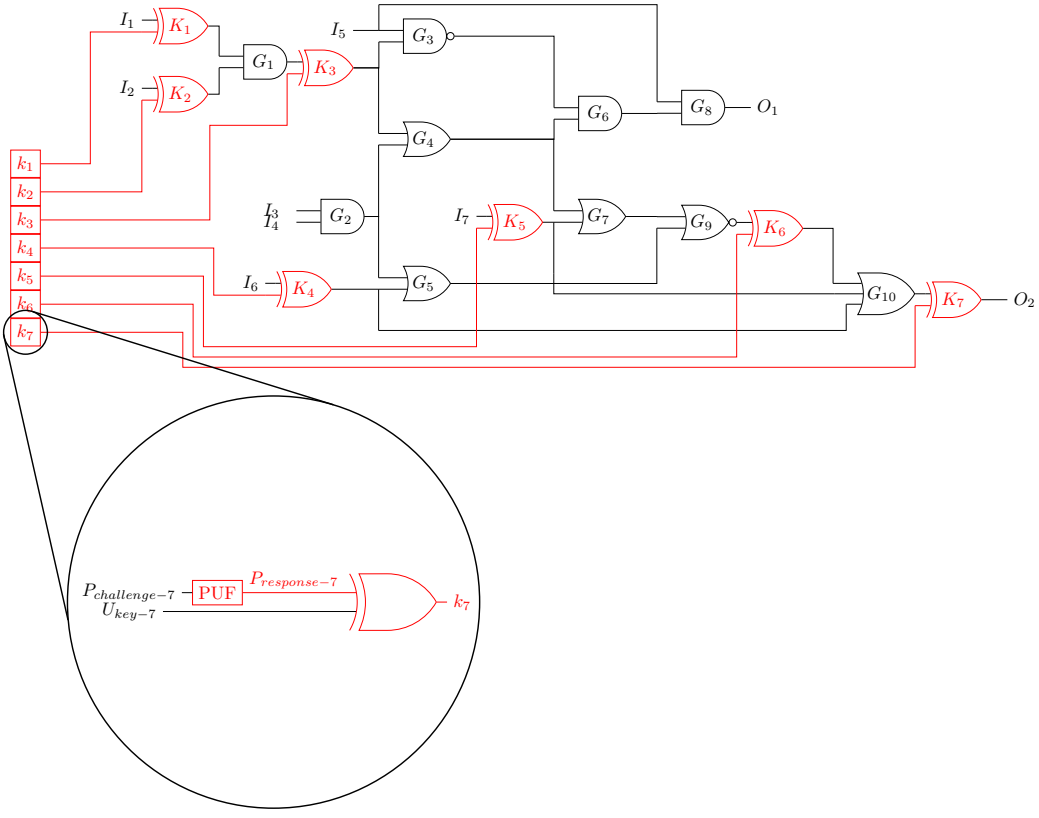
Plusieurs approches ont été étudiées, portant sur la position d'insertion et les types de portes logiques utilisées comme portes clés. Nous présentons ci-dessous un récapitulatif historique de l'état de l'art. Nous divisons les méthodes en deux catégories : les méthodes *pré-SAT* et *post-SAT*, en fonction de leur résistance à l'attaque SAT, détaillée dans la Section 3.2.2.

3.2.1 Logic locking pré-SAT

La méthode de *logic locking* a été introduite en 2008 sous le nom $EPIC$ (*Ending Piracy of Integrated Circuits*) [RKM08]. Cette méthode consiste à ajouter des portes $XOR/XNOR$ dans le circuit à des positions choisies aléatoirement. De ce fait, elle a été renommée *Random Logic Locking : RLL* [YS17]. Les portes XOR et $XNOR$ agissent comme des $buffers$ ou $inverseurs$ en fonction du bit de clé associé, assurant ainsi que la logique du circuit est respectée si la bonne clé est utilisée. Cette méthode ne propose initialement aucune métrique pour évaluer la sécurité du circuit. Par conséquent, tous les verrouillages étaient alors considérés comme équivalents en



(a) Circuit



(b) Circuit verrouillé

Figure 3.1. : Exemple de verrouillage logique avec des portes XOR en portes clés

termes de sécurité. Les travaux suivants montrent que cette assertion est incorrecte et proposent différentes métriques de sécurité pour améliorer cette approche.

Une première modification consiste à changer la nature de la porte clé en utilisant une table de vérité (LUT), qui est une fonction reprogrammable. Les auteurs de [BTZ10] discutent de la difficulté de retrouver la clé et proposent d'utiliser la distance de Hamming comme métrique de sécurité. Une distance de Hamming proche de 50% évite la possibilité d'appliquer la théorie du calcul approximatif (*Approximate Computing*) [HO13]. En effet, une distance de Hamming de 0% signifie que le verrouillage ne change pas la logique du circuit avec une clé incorrecte, tandis qu'une distance de 100% indique que le circuit donne exactement l'inverse des résultats attendus. Dans ce dernier cas, il suffirait d'inverser les sorties pour obtenir un circuit fonctionnel.

Un bit de clé erroné dans une porte clé modifie le circuit localement. Le signal protégé par la porte clé est alors erroné comme une faute. Cette faute impacte un nombre de sorties dans le circuit. Les auteurs de [Raj+12a] proposent de mesurer l'impact d'une faute dans les sorties comme métrique de sécurité. Cette mesure est ensuite utilisée pour insérer une porte clé $XOR/XNOR$. Ainsi, chaque porte clé fausse impactera fortement les sorties. Une extension de ces travaux a été développée dans [Raj+15]. Les auteurs proposent d'insérer des MUX mélangeant deux signaux du circuit en utilisant une métrique de contradiction entre les signaux définie dans cet article. La méthode avec XOR permet d'obtenir des distances de Hamming proches de 50%. Cependant, la méthode avec MUX n'atteint pas les 50%.

Les auteurs de [Raj+12b] ont observé qu'un attaquant possédant la *netlist* verrouillée et une copie fonctionnelle du circuit, aussi appelée oracle, peut retrouver la clé d'un circuit verrouillé. Cette méthode est connue sous le nom d'attaque par sensibilisation (*sensitization attack*). Elle implique de projeter une valeur interne du circuit sur une sortie. Pour contrer cette attaque, ils suggèrent de réaliser un *logic locking* avec des portes $XOR/XNOR$ placées de manière à ce qu'elles interfèrent entre elles et compromettent l'efficacité de l'attaque. Ils introduisent alors une métrique pour évaluer la robustesse de la clé. Leur méthode, initialement nommée *Strong Logic Obfuscation*, a été renommée plus tard en *Strong Logic Locking* (SLL). Dans [Yas+16b], les auteurs ont peaufiné leur méthode en affinant la mesure de robustesse de la clé en fonction de l'effort théorique nécessaire pour la retrouver. Ils démontrent également que leur méthode résiste à l'attaque *hill climbing* [PM15]. Cette attaque débute avec une clé aléatoire et effectue des inversions de bits pour minimiser le nombre de bits de sortie divergents par rapport à une table d'entrées/sorties pré-calculée.

Une autre proposition consiste à mélanger les signaux d'un circuit en utilisant des *Wire Scrambling cells (WS-cells)* [ZJ13]. Ces cellules opèrent en coupant plusieurs signaux du circuit et en les entrelaçant à l'aide d'un MUX, contrôlé par une clé de permutation en entrée de sélection. Les auteurs ont affiné cette méthode dans des travaux ultérieurs [ZJ16a ; ZJ16b] et discutent de la résistance de leur méthode face à différentes attaques. En ciblant notamment des signaux rares, ils limitent la possibilité d'insérer un CTM, en s'appuyant partiellement sur les travaux de la méthode suivante. En outre, en entrelaçant les signaux, les cônes logiques sont également entremêlés, ce qui complique la réalisation d'une attaque par analyse de cône logique. Cette approche vise à réduire les efforts de calcul nécessaires pour retrouver une clé en isolant certaines parties du circuit. Les auteurs soutiennent également que cette méthode résiste à l'attaque SAT, que nous examinerons dans la Section 3.2.2.

Les méthodes de *logic locking* se sont principalement axées sur la solidité de la clé, ainsi que sur la corruption des sorties. Les auteurs de [Dup+14b] proposent une méthode qui cherche à limiter l'insertion de CTMs (Chevaux de Troie Matériels) en réduisant les signaux rares. Les signaux rares sont ceux qui sont difficilement contrôlables, avec peu de vecteurs d'entrée permettant de fixer leur valeur. Ces signaux constituent de parfaits candidats pour l'activation d'un CTM. Limiter ces signaux réduit donc les possibilités d'insertion d'un CTM. Les portes clés proposées dans cette méthode sont des portes AND et OR. Toutefois, si un attaquant possède la *netlist* verrouillée, il peut déduire la clé, car les bits de clé sont liés à la porte logique. Une première amélioration de cette méthode a été proposée [Sam+16], consistant à utiliser des portes XOR et $XNOR$ pour éliminer ces signaux rares. Par la suite, les auteurs de [Mar+17] ont introduit l'usage d'un algorithme évolutionnaire multi-objectif. Le premier objectif de cet algorithme est la minimisation de la taille de la clé, et donc du nombre de portes, impactant la surface du circuit. Le second objectif concerne la corruption des sorties, en s'appuyant sur les travaux de [Raj+12a] qui utilisent des fautes, comme mentionné précédemment. Enfin, le dernier objectif est de minimiser les signaux rares. Ces méthodes réduisent considérablement le nombre de signaux rares, limitant ainsi les possibilités d'insertion d'un CTM. Les améliorations apportées permettent d'atteindre une corruption proche de 50%. Cependant, la robustesse de la clé face à des attaques visant à la retrouver n'est pas abordée dans ces travaux.

Les attaques par sensibilisation sont effectuées sur toutes les sorties du circuit. Cependant, si la *netlist* du circuit est connue, un attaquant peut extraire le cône logique d'une sortie, déterminer quelle porte clé se trouve dans ce cône, et ainsi limiter la taille de l'attaque. Pour contrer cela, les auteurs de [LT15] proposent

d'ajouter des MUX en complément des XOR afin de mélanger plusieurs cônes logiques.

En s'appuyant sur les travaux de détection de [SS15], il est établi que plus un chemin est court, plus l'impact d'un CTM sur le délai du circuit est important, et donc plus il est détectable. Dans l'objectif de faciliter la détection des CTM, les auteurs de [NHB15] proposent d'insérer des $XOR/XNOR$ et des MUX pour minimiser le plus grand des plus courts chemins (*maximum shortest path*), réduisant globalement tous les plus courts chemins. Cette méthode, par la suite améliorée et renommée "ESCALATION" [NHB16 ; NHB18], a fait l'objet d'une étude sur la corruption des sorties, révélant une faible distance de Hamming, ce qui était attendu puisque la corruption n'était pas l'objectif principal. Toutefois, pour cette méthode, les auteurs n'ont pas discuté des attaques visant les clés. Dans la même logique, une méthode visant à faciliter la détection par analyse de la consommation électrique a également été proposée [Nej+19].

Jusqu'à présent, les méthodes de *logic locking* présentées utilisent les trois métriques suivantes :

1. la corruption des données, mesurée par la distance de Hamming
2. la robustesse de la clé face aux différentes attaques
3. la difficulté d'insertion CTMs, en se basant sur les signaux rares

Les deux premières métriques visent à renforcer le *logic locking* dans son objectif principal : la protection des propriétés intellectuelles (IPs). La troisième métrique représente une utilisation intéressante et détournée de la méthode, cherchant à protéger les circuits contre l'insertion de CTMs en plus de la protection des IPs.

Cependant, une attaque a bouleversé ces méthodes en permettant de retrouver les clés des circuits verrouillés. Cette attaque est connue sous le nom d'attaque SAT que nous présentons ci-dessous.

3.2.2 Attaque SAT

Les auteurs de [SRM15] ont évalué la sécurité des méthodes de *logic locking*. Ils ont introduit une nouvelle attaque, émanant d'une fonderie malveillante. Les attaquants, disposant des informations sur le *layout*, peuvent reconstruire la *netlist* du circuit [TJ09]. En outre, ils ont accès à un circuit fonctionnel, qui pourrait par exemple être acquis sur le marché. Cette attaque implique de modéliser le circuit avec des formulations booléennes pour créer un modèle de *Quantified Boolean Formula (QBF)*.

Grâce à une approche itérative et en exploitant la puissance de calcul des solveurs SAT, cette méthode s'avère extrêmement efficace pour récupérer les clés.

Pour modéliser le problème en tant que QBF, on commence par représenter le circuit verrouillé sous la forme d'une formule logique $C\left(\overrightarrow{X}, \overrightarrow{K}, \overrightarrow{Y}\right)$. Notons que $\overrightarrow{X} \in \{0,1\}^M$, $\overrightarrow{K} \in \{0,1\}^L$, et $\overrightarrow{Y} \in \{0,1\}^N$, où M représente le nombre d'entrées du circuit, L la taille de la clé, et N le nombre de sorties. La formule est valide si $\overrightarrow{Y}$ correspond à la sortie du circuit lorsque $\overrightarrow{X}$ est l'entrée et $\overrightarrow{K}$ la clé utilisée. Nous cherchons alors à satisfaire la contrainte donnée par l'équation (3.1).

$$F_1 = C\left(\overrightarrow{X}, \overrightarrow{K_1}, \overrightarrow{Y_1}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_2}, \overrightarrow{Y_2}\right) \tag{3.1}$$

Cette formule, pour une même entrée $\overrightarrow{X}$, permet d'obtenir deux sorties différentes, $\overrightarrow{Y_1}$ et $\overrightarrow{Y_2}$, avec leurs clés respectives $\overrightarrow{K_1}$ et $\overrightarrow{K_2}$. Ces variables peuvent être égales ou différentes selon cette formulation. Ensuite, on cherche à résoudre le problème $is\_sat\left(F_1 \wedge \left(\overrightarrow{Y_1} \neq \overrightarrow{Y_2}\right)\right)$. Si la formule est satisfiable, alors il existe deux sorties différentes pour la même entrée, ce qui implique l'existence de deux clés distinctes produisant des résultats différents pour un même $\overrightarrow{X}$. Cela nous fournit une information importante : au moins l'une des deux clés est incorrecte. On peut alors utiliser le circuit fonctionnel pour déterminer quelle clé est erronée (ou si les deux le sont). On récupère l'assignation de l'itération i : $X_i \leftarrow \overrightarrow{X}$, puis on évalue cette entrée : $Y_i \leftarrow eval(X_i)$. Par la suite, on contraint les prochaines assignations à respecter cette entrée en ajoutant la contrainte (3.2). Cette méthode permet d'éviter de définir exhaustivement le circuit fonctionnel, ce qui peut s'avérer difficile en pratique.

$$F_{i+1} = F_i \wedge C\left(X_i, \overrightarrow{K_1}, Y_i\right) \wedge C\left(X_i, \overrightarrow{K_2}, Y_i\right) \tag{3.2}$$

On continue l'itération tant qu'il existe des clés ayant des sorties différentes pour une même entrée. Lorsque le problème devient insatisfiable, toute clé K_1 qui satisfait les contraintes supplémentaires déverrouille le circuit. On obtient alors l'Algorithme 1 : $delock(C)$.

Les auteurs de [SRM15] expliquent pourquoi cet algorithme est efficace, ce qui permet ensuite de comprendre les pistes de défense possibles. Ils définissent une classe d'équivalence de clés. On dit que $\overrightarrow{K_1} \equiv \overrightarrow{K_2}$ si et seulement si $\forall \overrightarrow{X} : C\left(\overrightarrow{X}, \overrightarrow{K_1}, \overrightarrow{Y}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_2}, \overrightarrow{Y}\right)$. Plutôt que de trouver la clé initialement prévue, l'Algorithme 1 cherche la classe d'équivalence de la clé secrète K_s. Pour être précis, à chaque itération, le solveur SAT trouve un vecteur d'entrée discriminant, appelé *Distinguish Input Pattern (DIP)*. Ce DIP permet d'éliminer une ou plusieurs classes d'équivalence à chaque itération. Notons qu'il existe au plus 2^L classes d'équivalence, avec une clé

Algorithme 1 : $delock(C)$

Entrées : C : le circuit combinatoire verrouillé
eval : une fonction d'évaluation du circuit fonctionnel
Résultat : Retourne une clé $\overrightarrow{K_c}$ déverrouillant le circuit

$i \leftarrow 1$
$F_1 \leftarrow C\left(\overrightarrow{X}, \overrightarrow{K_1}, \overrightarrow{Y_1}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_2}, \overrightarrow{Y_2}\right)$
tant que $is\_sat\left(F_i \wedge \left(\overrightarrow{Y_1} \neq \overrightarrow{Y_2}\right)\right)$ **faire**
    $X_i \leftarrow sat\_assignement_{\overrightarrow{X}}\left(F_i \wedge \left(\overrightarrow{Y_1} \neq \overrightarrow{Y_2}\right)\right)$
    $Y_i \leftarrow eval(X_i)$
    $F_{i+1} \leftarrow F_i \wedge C\left(X_i, \overrightarrow{K_1}, Y_i\right) \wedge C\left(X_i, \overrightarrow{K_2}, Y_i\right)$
    $i \leftarrow i + 1$
fin
$\overrightarrow{K_c} \leftarrow sat\_assignement_{\overrightarrow{K_1}}(F_i)$
return $\overrightarrow{K_c}$

différente dans chaque classe. L'efficacité de l'Algorithme 1 repose sur un nombre de classes d'équivalence relativement faible. Pour 90% des circuits testés, l'algorithme a trouvé une clé en moins de 250 itérations. Les auteurs proposent également une amélioration du temps de calcul par une recherche segmentée sur des parties spécifiques du circuit.

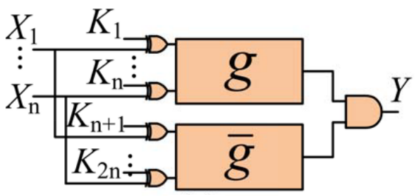
Pour se défendre contre cette attaque, il existe deux approches :

1. Augmenter le nombre d'itérations en limitant le nombre de clés dans chaque classe d'équivalence
2. Augmenter le temps de calcul requis pour chaque itération

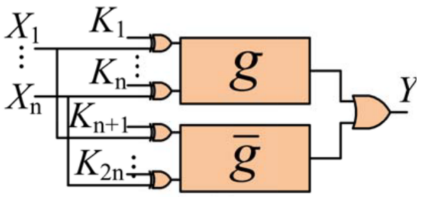
3.2.3 Logic locking post-SAT

Pour augmenter le temps de calcul nécessaire à chaque itération, les auteurs de [Yas+16b] suggèrent d'utiliser un bloc de cryptographie AES pour coder une partie de la clé, rendant ainsi l'exécution de l'attaque SAT plus difficile. Cependant, ils notent que les blocs AES étant des structures connues, ils peuvent être identifiés et retirés. Une fois retirés, il est alors possible d'appliquer l'attaque SAT sur le reste de la clé.

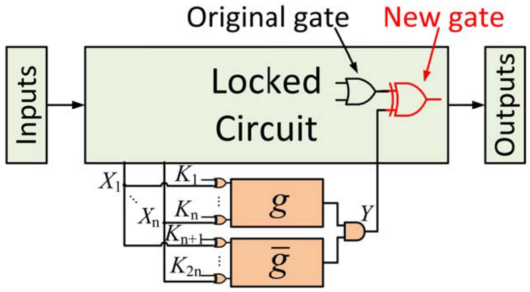
Pour augmenter le nombre d'itérations, les auteurs de [Yas+16a] soutiennent que si, pour chaque entrée, une seule clé erronée est possible, chaque DIP ne cassera qu'une seule clé à chaque itération. Une telle méthode garantit donc un nombre



(a) Type-0 Anti-SAT block



(b) Type-1 Anti-SAT block

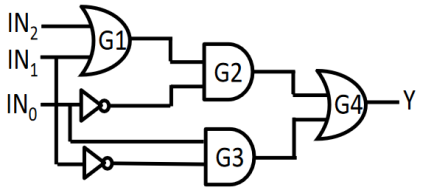


(c) Circuit verrouillé

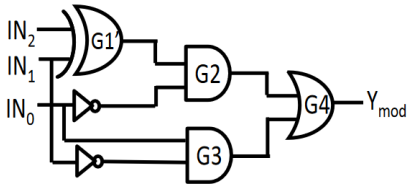
Figure 3.2. : Exemple de verrouillage anti-SAT [XS19]

d'itérations exponentiel pour l'Algorithme 1. Dans cette optique, ils proposent une méthode nommée $SARLock$. Ils introduisent une fonction ponctuelle *(point function)*, qui ne renvoie 1 que pour une sortie spécifique et 0 dans tous les autres cas. Cette fonction garantit qu'une seule entrée, composée d'un vecteur d'entrée et d'un vecteur clé, soit incorrecte. Ainsi, chaque DIP (qui est un vecteur d'entrée) ne discrimine qu'une seule clé. Cependant, cette méthode génère peu d'erreurs, ce qui signifie qu'une clé incorrecte produit un circuit presque correct. Les auteurs reconnaissent des faiblesses dans leur méthode face aux attaques par suppression (*removal attack*) ou par sensibilisation. Ils proposent alors d'hybrider $SARLock$ avec une méthode *pré-SAT* couvrant ces attaques, comme le SLL. La clé pour le SLL est ainsi calculée pour maximiser l'interférence, tandis que la clé pour le $SARLock$ est utilisée pour maximiser le nombre de DIP afin de lutter contre l'attaque SAT.

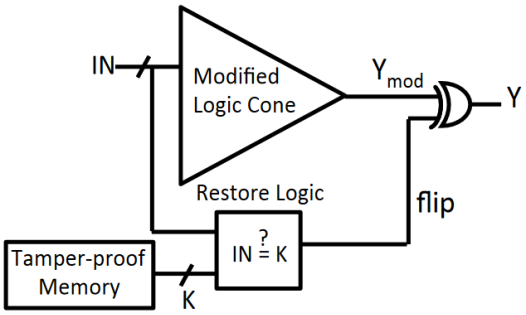
Dans le même objectif, les auteurs de [XS16] proposent d'utiliser des blocs *anti-SAT* avec une implémentation légère. La Figure 3.2 illustre cette méthode. La Figure 3.2a présente un bloc anti-SAT de type 0, où, si la clé est correcte, le circuit retourne 0. La Figure 3.2b montre un bloc anti-SAT de type 1, où, si la clé est correcte, le circuit retourne 1. Le bloc g est une porte AND à n entrées et le bloc $\overline{g}$ est une porte $NAND$ à n entrées. La Figure 3.2c montre l'insertion de ce bloc dans un circuit. Les entrées du bloc sont des signaux internes du circuit, et la sortie du bloc est connectée à une porte $XOR/XNOR$ ajoutée dans le circuit. La construction de ce bloc est légère et efficace contre les attaques SAT, car chaque DIP ne discrimine qu'une clé



(a) Exemple de cône logique



(b) Exemple de cône logique modifié



(c) Implémentation de $TTLock$

Figure 3.3. : Exemple de verrouillage $TTLock$ [Yas+17b]

au maximum. Les auteurs recommandent également d'utiliser une méthode *pré-SAT* en complément pour aborder d'autres objectifs que la seule résistance à l'attaque SAT. Dans une évolution de ces travaux [XS19], les auteurs suggèrent d'utiliser le SLL comme méthode *pré-SAT*. Ils proposent aussi une solution face à l'attaque par suppression, qui consiste à retirer le bloc *anti-SAT*. Comme ce bloc est externe au circuit principal, il peut être identifié et supprimé. Pour contrer cela, les auteurs suggèrent de masquer les signaux d'entrée X avec des MUX reliés à une troisième clé.

Les auteurs de [Yas+17b] proposent une autre méthode appelée $TTLock$. Cette méthode sélectionne un cône logique, comme illustré dans la Figure 3.3a, puis inverse la sortie du cône pour un unique vecteur d'entrée. Suite à une resynthèse du cône, comme montré dans la Figure 3.3b, celui-ci est correct pour tous les vecteurs d'entrée sauf un : 110. Il est alors nécessaire d'inverser le signal de sortie du cône pour cette entrée, et la clé du *logic locking* discrimine ce vecteur, comme illustré dans la Figure 3.3c. Cette méthode résiste aux attaques par sensibilisation, à l'attaque SAT, ainsi qu'à l'attaque par suppression.

Suite à l'émergence de ces méthodes de *logic locking*, deux nouvelles attaques ont été développées. Le problème d'attaque par oracle peut être considéré comme un problème de *machine learning*, spécifiquement un cas d'*active learning* [Set09]. Les auteurs de [Sha+17a] proposent alors une attaque approchée nommée $AppSAT$.

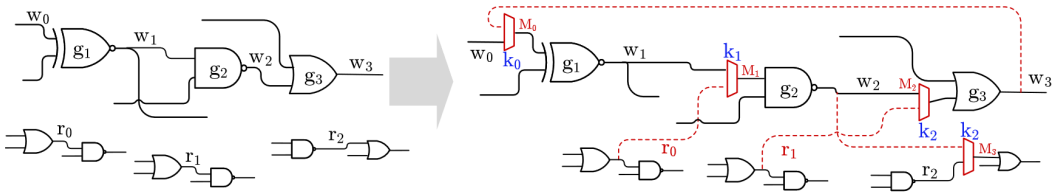


Figure 3.4. : Exemple de *logic locking* cyclique [Sha+17b]

Cette attaque s'inspire du paradigme *probably-approximately-correct*, où l'attaque fournit une $\epsilon - approximation$ du circuit avec une probabilité λ. L'attaque permet ainsi de retourner une clé ayant une faible corruption ou une clé correcte. Selon les résultats, $AppSAT$ peut trouver les clés des méthodes *pré-SAT*, avec ou sans l'ajout de méthodes *post-SAT*. Elle permet notamment de retrouver la clé SLL dans un verrouillage combinant $Anti\text{-}SAT$ et SLL. Toutefois, pour les méthodes *post-SAT*, les performances de l'attaque $AppSAT$ ne sont pas supérieures à celles de l'attaque SAT.

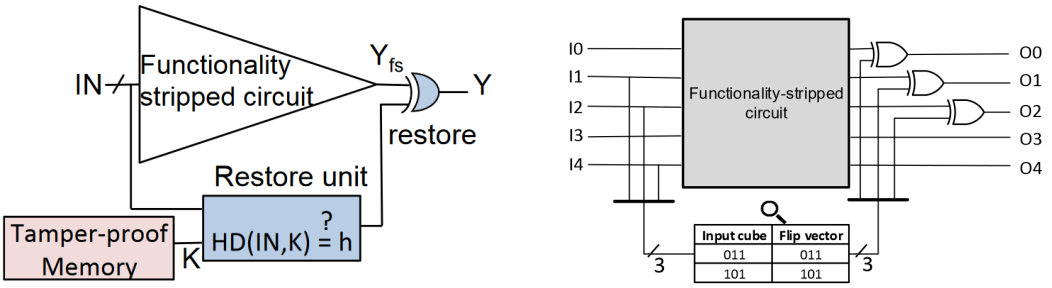
La deuxième attaque est une extension de l'attaque SAT. L'attaque *double-DIP* $(2DIP)$ [SZ17] vise à trouver un DIP qui discrimine deux clés à la fois, réduisant ainsi le nombre d'itérations nécessaires. La formule à satisfaire devient alors :

$$F_1 = C\left(\overrightarrow{X}, \overrightarrow{K_1}, \overrightarrow{Y_1}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_2}, \overrightarrow{Y_2}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_3}, \overrightarrow{Y_1}\right) \wedge C\left(\overrightarrow{X}, \overrightarrow{K_4}, \overrightarrow{Y_2}\right) \wedge \left(\overrightarrow{Y_1} \neq \overrightarrow{Y_2}\right) \wedge \left(\overrightarrow{K_1} \neq \overrightarrow{K_3}\right) \wedge \left(\overrightarrow{K_2} \neq \overrightarrow{K_4}\right)$$

Les résultats montrent que cette attaque permet de trouver rapidement les clés de la méthode $SARLock$ combinée à un verrouillage *pré-SAT*, à l'exception de la combinaison $SARLock + SLL$, qui requiert toujours un temps exponentiel. Cependant, en fixant une clé arbitraire pour le $SARLock$ (par exemple $\overrightarrow{0}$), qui produit alors des résultats corrects pour presque toutes les entrées, il devient possible de retrouver la clé du SLL en moins de 100 secondes. Ainsi, cette méthode permet de découvrir une clé *pré-SAT* même en présence d'une hybridation *post-SAT*.

Ces attaques montrent qu'il est possible de réduire les stratégies de *logic locking* multicouches à des méthodes *post-SAT*, simplifiant ainsi la complexité des attaques.

Pour limiter l'efficacité de l'attaque SAT, les auteurs de [Sha+17b] proposent d'ajouter des signaux rétroactifs dans le circuit. Dans la Figure 3.4, le signal w_3 est renvoyé dans un MUX M_0 lié à un bit de clé k_0. Cette méthode mélange plusieurs signaux pour compliquer la compréhension de la logique du circuit. Les



(a) Exemple d'un $SFLL\text{-}HD$ **(b)** Exemple d'un $SFLL\text{-}flex$

Figure 3.5. : Exemple de verrouillage $SFLL$ [Yas+17a]

auteurs proposent une protection théorique contre l'attaque SAT, mais sans fournir d'expérimentations pratiques pour étayer cette affirmation.

Face à cette approche de *logic locking* cyclique, une évolution de l'attaque SAT, nommée $CycSAT$ [ZJK17], a été proposée. $CycSAT$ part du principe qu'un circuit déverrouillé ne contient pas de cycle. Elle intègre une condition dans le solveur SAT assurant que chaque clé proposée aboutisse à un circuit acyclique. Les résultats de l'article montrent que l'attaque SAT est peu efficace en pratique contre les méthodes de *logic locking* cyclique. En revanche, l'attaque $CycSAT$ s'avère capable de retrouver les clés de presque tous les circuits testés.

Les auteurs de [Yas+17a] proposent d'améliorer le $TTLock$ en adoptant le principe de cône logique modifié, aboutissant à la méthode *Stripped-Functionality Logic Locking (SFLL)*. Cette méthode se décline en deux variantes : $SFLL\text{-}HD$ et $SFLL\text{-}flex$. Le $SFLL\text{-}HD$ est conçu pour un verrouillage générique, alors que le $SFLL\text{-}flex$ cible la protection d'une fonctionnalité spécifique. Le $SFLL\text{-}HD$ inverse le cône logique pour tous les vecteurs d'entrée à une distance de Hamming h de la clé. Cette approche protège $\binom{k}{h}$ vecteurs d'entrée dans un cône. Comme le montre la Figure 3.5a, le module HD inverse les sorties du cône modifié (*Functionality stripped circuit*) pour les entrées à une distance de Hamming h de la clé. On note que pour $h = 0$, le $SFLL\text{-}HD$ est équivalent au $TTLock$. En augmentant h, $SFLL\text{-}HD$ renforce la sécurité de $TTLock$ avec un surcoût de surface modéré. Le $SFLL\text{-}flex$ emploie un $tamper\ proof\ LUT$ [Tuy+06] pour protéger c vecteurs d'entrée critiques de longueur k, choisis par le concepteur, associés à un vecteur d'inversion (*flip vector*). Dans la Figure 3.5b, le circuit est modifié pour inverser les sorties pour les entrées sélectionnées, en fonction du $flipvector$ correspondant. Cette méthode résiste aux attaques SAT, aux attaques par sensibilisation, et aux attaques par suppression. De plus, n'ayant qu'une seule couche de *logic locking*, les attaques multicouches n'apportent aucun bénéfice.

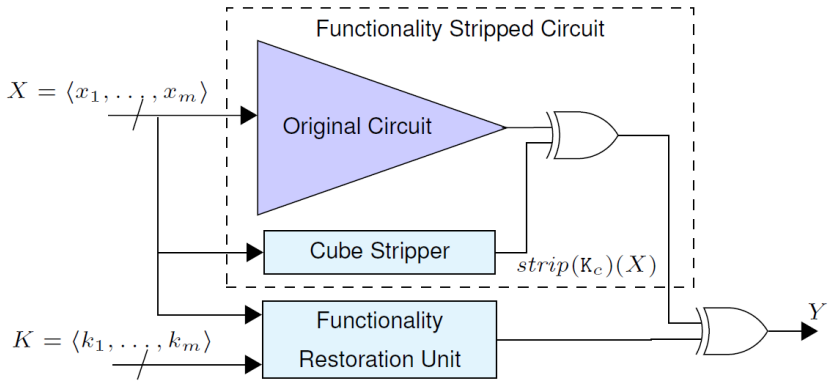


Figure 3.6. : Exemple d'un *logic locking* avec un cône logique modifié [SS19]

Les auteurs de [Sen+18] ont développé une automatisation du $SFLL\text{-}flex$ nommée $SFLL\text{-}fault$. Cette méthode vise à sélectionner le cône logique qui minimise la surface, en analysant les vecteurs d'entrée avec un $ATPG$ et en garantissant un niveau de sécurité minimal. Cette approche est entièrement automatique et réduit la surface utilisée de 30%.

Une attaque a été proposée contre les *logic locking* qui utilisent des *functionality stripped circuits* et des *restoration units* comme le $TTLock$ et le $SFLL\text{-}HD$ [SS19]. Elle est nommée attaque $FALL$ (*Functional Analysis attacks on Logic Locking*). La Figure 3.6 est un exemple de circuit attaquable. On distingue le cône modifié (*functionality stripped circuit*) composé du cône logique original (*original circuit*) et d'un bloc de confusion (*cube stripper*). Pour ne pas perdre en fonctionnalité, il y a le bloc de restauration (*restoration unit*). Cette attaque consiste à identifier les blocs de confusion, puis d'extraire les vecteurs d'entrée protégés. Ensuite, en utilisant ces vecteurs, retrouver la clé numérique. Lorsque la valeur h est inférieure à $\frac{k}{4}$, l'attaque trouve les clés de tous les circuits. Pour une valeur $h = \frac{k}{3}$, moins de la moitié des clés ont été trouvées. Cette attaque démontre une faiblesse pour des valeurs de h faibles, et un angle d'attaque meilleur pour des valeurs de h grandes.

Les auteurs de [Liu+20] proposent une amélioration de $Anti\text{-}SAT$ appelée *Strong Anti SAT* (SAS). Ils constatent que les *logic locking* utilisant des fonctions ponctuelles, en insérant une fausse clé, ne possèdent que peu de vecteurs d'entrée (souvent un seul) qui produisent un résultat faux. Les auteurs identifient des vecteurs d'entrée critiques, qui impactent beaucoup la logique du circuit. Ils ajoutent alors une fonction H qui va forcer un groupe de fausses clés à inverser le signal du circuit. Ceci permet alors d'ajouter de la corruption dans le circuit. On note qu'aucun détail supplémentaire n'est donné sur l'implémentation de ces blocs H dans l'article. En limitant ce nombre de vecteurs critiques, on garde un facteur exponentiel pour la

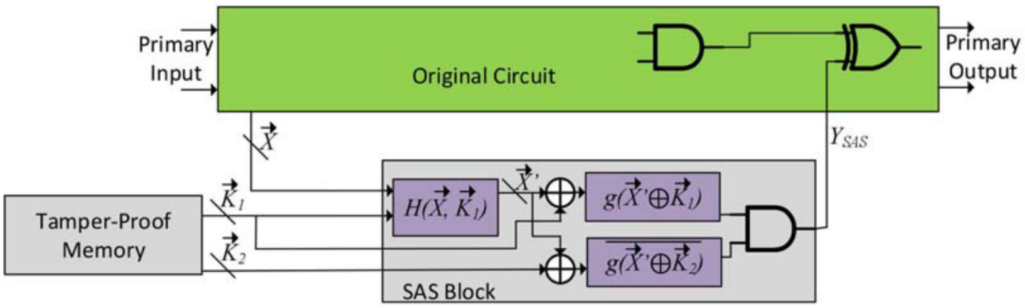


Figure 3.7. : Exemple d'un *strong anti-SAT logic locking* avec un bloc SAS [Liu+20]

sécurité, ainsi qu'une corruption efficace. Dans la Figure 3.7, on observe que les entrées $\vec{X}$ sont transformées en $\vec{X'} = H\left(\vec{X}, \vec{K_1}\right)$. Si $\vec{X}$ est un vecteur critique, alors la fonction H s'assure que $g\left(\vec{X'}, \vec{K_1}\right) = 1$, sinon $\vec{X'} = \vec{X}$. Ainsi, cela permet au bloc SAS d'obtenir un grand impact sur le circuit, en gardant une sécurité exponentielle. Le nombre d'itérations SAT nécessaire pour retrouver la clé est supérieur au $SFLL\text{-}flex$. Les résultats montrent une meilleure sécurité pour une plus faible surface.

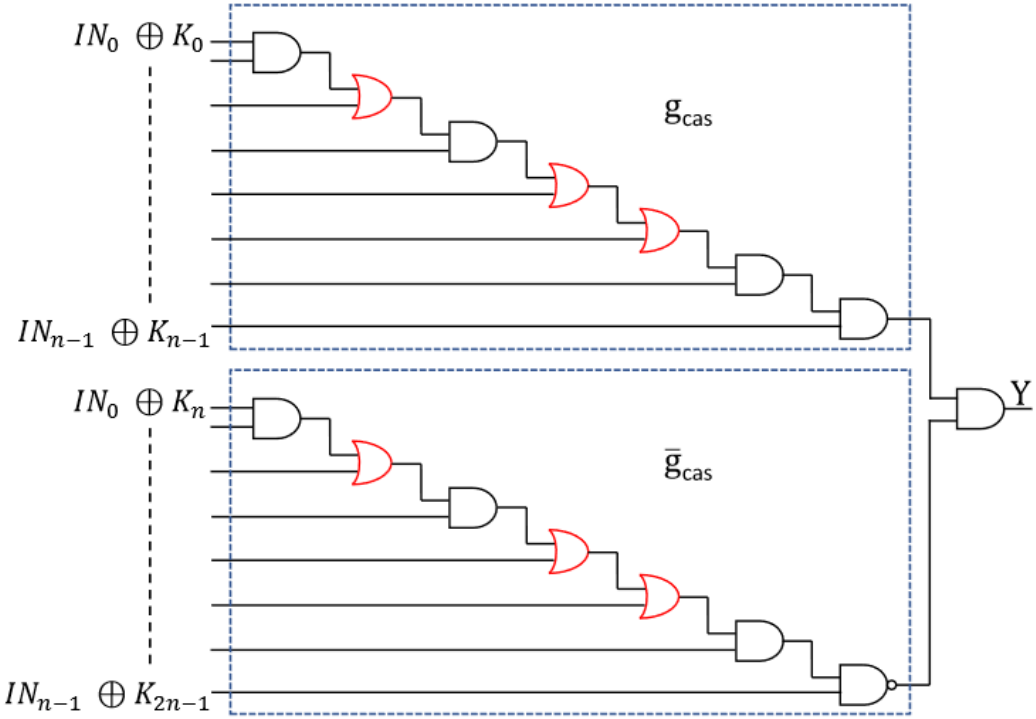


Figure 3.8. : Exemple d'un bloc $CAS\text{-}Lock$ [Sha+19]

Une autre amélioration d'$Anti\text{-}SAT$ est proposée par les auteurs de [Sha+19] appelée *Cascaded Locking* $CAS\text{-}Lock$. Cette méthode change les blocs $Anti\text{-}SAT$ qui sont des arbres de AND et $NAND$ en des cascades de AND et OR, représentées dans la Figure 3.8. Nous retrouvons les blocs complémentaires g_{cas} et $\overline{g}_{cas}$. Ce verrouillage

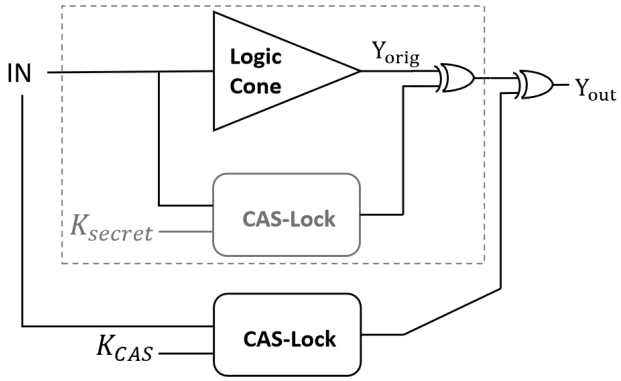


Figure 3.9. : Exemple d'un $M\text{-}CAS\text{-}Lock$ [Sha+19]

permet de se prémunir des attaques *black box* (l'attaquant n'a connaissance que de la *netlist*), comme l'attaque SAT ou $AppSAT$. Cependant, elle ne permet pas de se prémunir des attaques *white box* (l'attaquant connaît la *netlist*), comme l'attaque par suppression. Ainsi, pour s'en prémunir, les auteurs proposent d'appliquer un SLL sur les blocs $CAS\text{-}Lock$. Cependant, cette hybridation diminue la protection contre l'attaque $AppSAT$, pour laquelle 50% des clés ont été retrouvées contre 5% sans. Les auteurs proposent alors une extension de leur méthode appelée le *Mirrored CAS-Lock* ($M\text{-}CAS\text{-}Lock$). Cette méthode applique un premier $CAS\text{-}Lock$ au circuit, avec une clé K_{secret} écrite en dur dans le circuit. Ce cône logique est alors resynthétisé pour que la clé soit cachée dans le circuit par les optimisations de synthèse. Ensuite, un deuxième $CAS\text{-}Lock$, identique au premier, verrouille le cône modifié. Seule la clé K_{secret} peut déverrouiller le circuit. La Figure 3.9 illustre l'implémentation. Le choix de la clé K_{secret} impacte fortement la protection contre l'attaque SAT, avec le compromis entre la corruption et la protection à l'attaque SAT. D'après les résultats, cette méthode nécessite moins d'espace et d'énergie que la méthode $SFLL\text{-}HD$, mais décroît plus les performances. La méthode $CAS\text{-}Lock{+}SLL$ casse le compromis entre la corruption et l'attaque SAT, mais est, dans certains cas, sensible à l'attaque $AppSAT$. La méthode $M\text{-}CAS\text{-}Lock$ a donc la même faiblesse que le $SFLL\text{-}HD$ pour l'attaque SAT, mais résiste à l'attaque $FALL$.

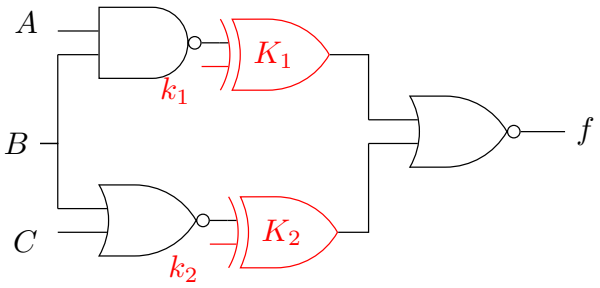
Très récemment, en s'appuyant sur les travaux d'un solveur SMT quantique [Lin+23], les auteurs de [BZ22] discutent de l'impact de l'informatique quantique sur les attaques SAT. L'accélération quadratique des solveurs SMT quantiques conduit à la conclusion qu'il est nécessaire de doubler la taille des clés pour maintenir un niveau de sécurité équivalent.

Une amélioration de l'attaque SAT est proposée dans [ZG23] accompagnée d'une analyse de son comportement. Les auteurs suggèrent d'analyser la logique du circuit

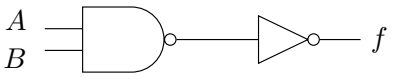
pour chaque DIP, afin de déterminer d'éventuelles relations entre les bits de clé pour ce DIP. Ces relations sont ensuite ajoutées aux contraintes du modèle SAT, ce qui permet de réduire l'espace de recherche des clés. Cette analyse permet de considérablement réduire le nombre d'itérations, malgré un surcoût temporel à chaque itération. Les auteurs analysent l'efficacité de l'attaque SAT sur le SLL, en argumentant qu'une mauvaise clé impacte plusieurs cônes logiques, ce qui affecte plusieurs sorties. Cela permet de casser l'interdépendance entre les clés. Ainsi chaque DIP discrimine un grand nombre de clé. Cependant, le SLL analysé ici est la première proposition issue de [Raj+12b] et non l'amélioration de [Yas+16b]. Les auteurs ont affiné la mesure théorique ainsi que les relations d'interférence, de sorte que l'argument présenté nécessite une réévaluation. En effet, par définition, deux clés en interférence ne peuvent être séparées dans deux cônes indépendants. Par conséquent, proposer une mesure précise ainsi qu'une optimisation optimale du SLL pourrait apporter une sécurité accrue contre l'attaque SAT. Les auteurs proposent également une analyse des *logic locking* utilisant des fonctions ponctuelles, comme $Anti\text{-}SAT$, $CAS\text{-}Lock$, et $SFLL\text{-}HD$. Ils prétendent pouvoir retrouver la clé après une itération en insérant une fausse clé partielle. Cette analyse n'a pas été menée sur le SAS.

Les attaques avec oracle, comme l'attaque par sensibilisation et l'attaque SAT, reposent sur la possibilité de facilement appliquer un vecteur d'entrée au circuit. Pour ce faire, les attaquants utilisent la chaîne de scan *(scan-chain)* pour forcer une valeur de registre et tester la logique combinatoire. En verrouillant cette chaîne de scan, il devient difficile, voire impossible, d'insérer des vecteurs de test. Certains auteurs proposent alors une méthode pour verrouiller la chaîne de scan, puis ajouter un SLL au circuit. Cette méthode s'appelle $kt\text{-}DFS$ (*key-trapped design for security*) [Kam+20]. Les résultats montrent que l'augmentation de la surface est raisonnable, inférieure à 7% sur les circuits testés. À notre connaissance, aucune attaque n'a été proposée contre cette méthode pour le moment.

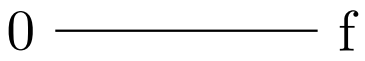
Des attaques sans oracle ont été proposées. Elles reposent sur des analyses structurelles des circuits. L'attaque par dé-synthèse [Mas+17] propose une attaque de recherche locale, similaire à l'attaque par *hill climbing*, pour les verrouillages *pré-SAT*. En observant que ces verrouillages modifient peu le circuit, en fixant une clé dans la *netlist* et en re-synthétisant, on obtient un second circuit. Si le circuit est très différent du premier, alors la clé est probablement incorrecte. En se basant sur le fait qu'un circuit est optimisé pour être simplifié avant l'ajout du verrouillage. Partant d'une clé, qui peut être aléatoire, l'attaque teste toutes les clés à une distance de Hamming de 1 qui minimisent la différence, jusqu'à ne plus trouver d'amélioration. Un exemple est donné ci-dessous.



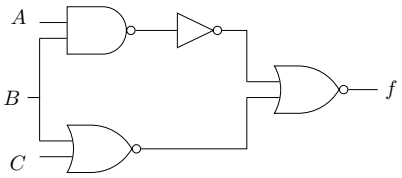
(a) Circuit verrouillé



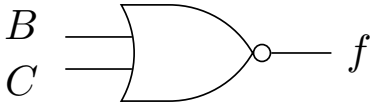
(b) Circuit re-synthétisé avec la clé 00



(c) Circuit re-synthétisé avec la clé 01



(d) Circuit re-synthétisé avec la clé 10



(e) Circuit re-synthétisé avec la clé 11

Figure 3.10. : Exemple d'attaque par dé-synthèse [Mas+17]

Example 3.2.2 (*Attaque par dé-synthèse*) : *Les auteurs de Figure 3.10 proposent un exemple. En partant d'un circuit verrouillé Figure 3.10a, le circuit peut être re-synthétiser en imposant une valeur aux bits de la clé. Dans cet exemple, le circuit final le plus proche du circuit verrouillé est celui de la clé* 10 *dans la Figure 3.10d. En se basant sur les arguments précédents, il est probable que la clé* 10 *soit la clé de déverrouillage. Les expérimentations ont été menées sur le* RLL *et ont permis de trouver au moins* 75% *des bits de clé dans* 50% *des expérimentations. Les auteurs supposent que cette attaque s'étend aux méthodes pré-SAT, mais aucune expérimentation ne vient étayer cette hypothèse.*

Dans les discussions de [Kam+22] (Section 5.2), les auteurs proposent de voir le problème du *logic locking* comme un problème de cryptographie appelé *zero-knowledge-proof (ZKP)*. Les concepteurs cherchent à partager un circuit ou une IP avec d'autres concepteurs ou des fonderies, sans divulguer la connaissance de la conception. Les auteurs argumentent que les *designs*, qu'ils soient verrouillés ou non, doivent être indiscernables, ce qui n'est pas le cas des verrouillages basés sur des fonctions ponctuelles ($SFLL$, $CAS-Lock$, etc.). De plus, le verrouillage doit être uniforme dans tout le circuit.

Les modèles d'attaques présentés utilisent une *netlist* ainsi qu'un oracle. Reconstruire une *netlist* est en soi une tâche complexe, et obtenir un oracle n'est pas toujours simple. Ces modèles d'attaques sont réalisables, mais demandent donc des efforts importants pour être mis en œuvre.

Après avoir présenté les méthodes existantes, nous constatons que plusieurs approches ont été proposées, avec des objectifs différents : maximiser la corruption, renforcer la résistance aux différentes attaques et lutter contre les chevaux de Troie matériels. À notre connaissance, $kt\text{-}DFS + SLL$ [Kam+20] proposent une protection efficace contre toutes les attaques connues, pour des coûts de surface raisonnables. À notre connaissance, il n'existe pas d'attaque structurelles ou comportementale contre le SLL, et l'association avec $kt\text{-}DFS$ empêche toute attaque par oracle.

3.3 Synthèse

De nombreuses approches différentes de *logic locking* ont été proposées. Nous remarquons que des méthodes apportent des modification pleinement intégrées au circuit et d'autres ajoutent des blocs logiques partiellement séparés du circuit original. Les approches intégrés au circuit correspondent aux méthodes *pré-SAT*. Les méthodes *post-SAT* ajoutent des blocs logiques partiellement séparés du circuit original. Cette séparation du circuit apporte des nouvelles attaques. Parmi les méthodes intégrés au circuit, le SLL est l'approche la plus élaborées en proposant une mesure de résistance pour retrouver la clé. Elle est la seul méthode *pré-SAT* utilisée par des méthodes *post-SAT* voulant verrouiller le circuit en plus des blocs logiques ajoutés au circuit.

Les auteurs de [Yas+16b] mettent en évidence que la position d'insertion de portes clés $XOR/XNOR$ modifie la sécurité et le chemin critique, donc les performances du circuit. Mais changer de positions modifient peu la consommation et la surface, le facteur déterminant est le nombre de portes clés.

Ces contre-mesures sont difficiles à mettre en œuvre sans expertise. Comme le font remarquer les auteurs de [Kne+20], il y a un besoin d'outil de conception intégrant des mesures de sécurités. Des outils automatiques permettent à un large nombre de concepteur de les implémenter dans leurs circuits. À notre connaissance, il n'existe pas d'outil intégrant des méthodes de *logic locking* dans un flot de conception, mais des entreprises s'y intéresse [Syn ; Syn23].

Nous avons remarqué que le SLL est une méthode *pré-SAT* qui est utilisée par de nombreuses méthodes *post-SAT*. Nous allons la présenter en détail dans la section suivante.

3.4 Strong logic locking

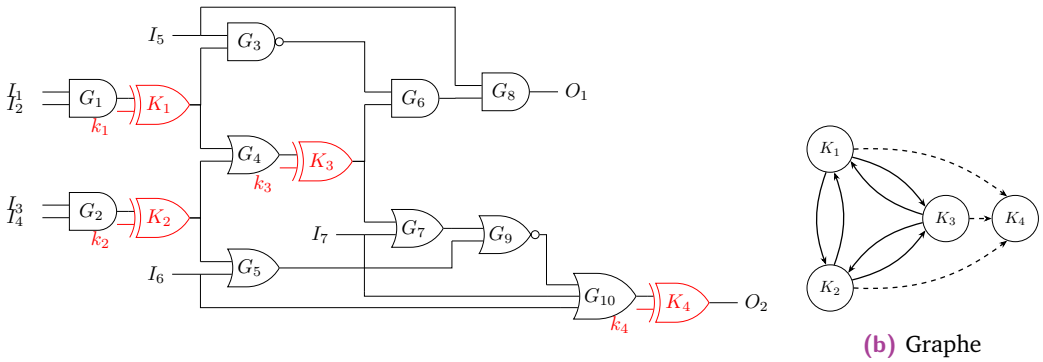
Le SLL [Raj+12b ; Yas+16b] est une méthode *pré-SAT* de *logic locking* qui vise à se prémunir contre une attaque par sensibilisation. Les auteurs définissent des relations binaires de muselage (Définition 3.4.1), d'interférence (Définition 3.4.2), et de *pairwise secure* (Définition 3.4.3) entre des portes clés. Ces informations sont synthétisées dans un graphe composé de portes clés pour sommets et d'arcs de muselage et d'interférence. La métrique de sécurité mesure le nombre d'arcs d'interférence dans le graphe d'interférence. Ils montrent que l'effort de l'attaquant est exponentiel en la taille de la plus grande clique (Définition 2.1.13) composée d'arcs d'interférence. Nous présentons les définitions ci-dessous, accompagnées d'un exemple. Pour optimiser le nombre de relation d'interférence, les auteurs proposent un algorithme glouton. En partant d'un nombre de portes clés initiales, choisi aléatoirement ou judicieusement, les auteurs vont chercher la position d'insertion d'une nouvelle porte clé qui va maximiser le nombre de relation d'interférence.

Définition 3.4.1 (*Muselage*) : *une **porte clé** est **muselable**, lorsqu'il existe un **vecteur d'entrée** permettant d'**inhiber** l'influence logique de son bit clé associé, pour déterminer le bit clé d'une **autre porte clé**.*

Définition 3.4.2 (*Interférence*) : *une **porte clé** K **interfère** avec une **autre porte clé** K', lorsque K n'est pas muselable pour K'.*

Définition 3.4.3 (*Pairwise secure*) : *deux **portes clés** K et K' sont **pairwise secure** si K interfère avec K' et inversement.*

Example 3.4.1 (*Exemple d'un SLL avec son graphe d'interférence*) : *Dans la Figure 3.11, nous avons un circuit verrouillé Figure 3.11a et le graphe d'interférence associé Figure 3.11b. Les arcs pleins représentent des relations d'interférence et les arcs en pointillés sont des relation de muselage. Lorsqu'il y a deux arcs pleins en aller-retour*



(a) Circuit verrouillé

(b) Graphe d'interférence

Figure 3.11. : Exemple de verrouillage logique avec le graphe d'interférence associé [Raj+12b]

entre deux sommets, nous avons par définition une relation pairwise secure. Les cliques sont alors composées de relation pairwise secure.

3.5 Conclusion

Le *logic locking* est une méthode apparue il y a 15 ans qui propose de lutter contre la surproduction et les CTM. Nous avons présenté différentes propositions de cette méthode. Elle a connue beaucoup d'améliorations suite à la découverte de différentes attaques. Nous avons consaté que la méthode de *Strong Logic Locking* est utilisée en addition de nombreuses méthodes *post-SAT*. Elle est intrégrée au circuit et apporte un obscurcissement uniforme, ce qui la rend robuste aux attaques par supression. À notre connaissance, aucune méthode de *logic locking* n'a été implémentée dans un outil EDA. Selon nous, le SLL est un candidat intéressant à intégrer dans un flot de conception pour limiter l'insertion des CTMs, limiter la surproduction et le vol d'IP.

De plus, nous avons également remarqué que, dans les méthodes de *pré-SAT*, les positions d'insertions des portes clés $XOR/XNOR$ déterminent la sécurité ainsi que l'impact sur les délais. Nous pouvons alors optimiser l'insertions de ces portes clés pour obtenir une sécurité satisfaisante en limitant l'impact du délai. Pour effectuer de telles optimisations, nous avons besoin de formules analytiques afin de mesurer la sécurité et l'impact sur le délai. La méthode de SLL optimise la sécurité avec un algorithme glouton, qui est une heuristique. Pour trouver de bons compromis, nous avons besoins de formules analytiques et d'algorithme de résolutions exacte pour le problème.

Dans cette thèse, nous souhaitons alors optimiser la méthode de SLL pour pouvoir l'intégrer dans des outils automatiques pour un flot de conception. Nous voulons modéliser l'insertion de portes clés du SLL sous forme de modèle mathématique afin de proposer des algorithmes de résolution optimaux. Le prochain chapitre est consacré à la modélisation du problème de sécurité, et à la proposition d'algorithmes de résolution efficaces.

4 Modélisation pour l'optimisation de la sécurité du Logic Locking

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous avons identifié le *SLL* comme une méthode intéressante à optimiser. Les auteurs de [Yas+16b] montrent que la taille et la consommation du circuit dépendent du nombre de portes ajoutées, tandis que le délai est très impacté par le choix des positions d'insertions. Maximiser la sécurité en minimisant l'impact sur le délai dépend des positions d'insertions des portes clés. Dans ce chapitre, nous allons proposer une nouvelle approche de résolution pour le

SLL, accompagnée d'une redéfinition plus fine du calcul de la sécurité. Nous définissons ce problème de sécurité, montrer les liens et les différences avec le problème de partition en cliques d'un graphe. Nous allons montrer que notre problème est $NP\text{-}DUR$ dans le cas général. Nous modélisons ensuite le problème de sécurité, puis proposons une deuxième modélisation en pré-calculant la liste des cliques maximales. Nous comparons ensuite ces deux modèles avec des résultats numérique. Nous cloturons sur la non-approximabilité du problème.

4.2 Nouvelle approche de résolution du SLL

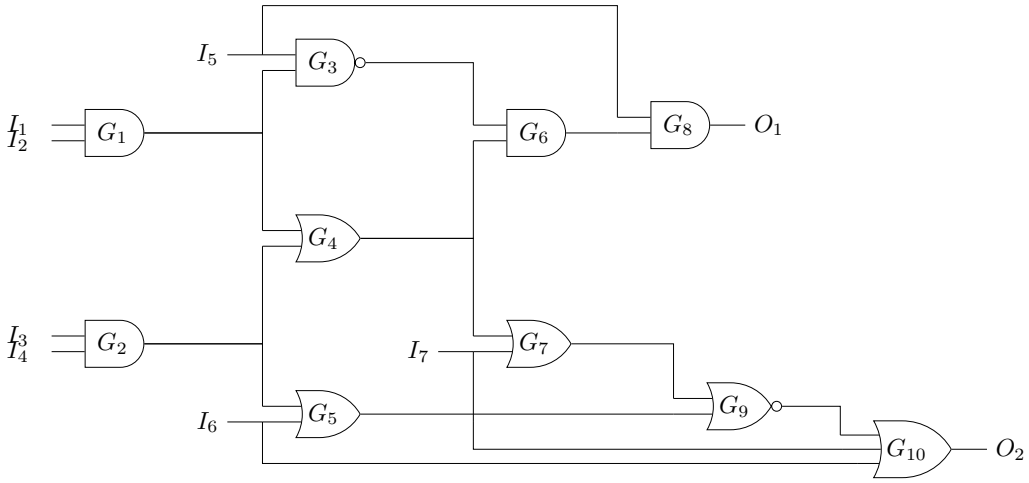
Les auteurs du *SLL* calculent le graphe d'interférence en même temps qu'ils choisissent des positions d'insertions des portes clés. Ces positions sont calculées avec un algorithme glouton en fonction de la contrôlabilité et de l'observabilité des portes. Cette approche peut se découper en trois étapes, à itérer jusqu'à obtenir une solution satisfaisante :

1. Trouver une nouvelle position d'insertion pour maximiser la sécurité
2. Calculer la relation de *pairwise secure* avec toutes les positions sélectionnées
3. Ajouter ce sommet dans le graphe d'interférence

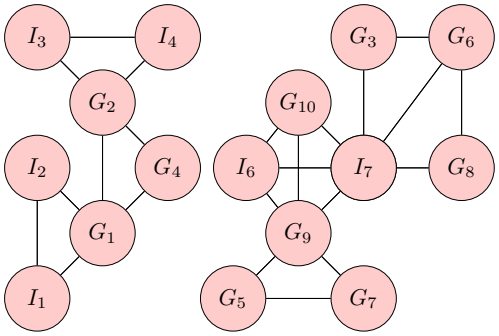
Cet algorithme apporte de bonnes solutions, mais nous voulons trouver les positions d'insertion de porte clés optimales pour la sécurité. Pour cela, nous devons connaître toutes les relations d'interférences. Nous proposons alors de modifier l'ordre des étapes comme suit :

1. Calculer toutes les relations de *pairwise secure*
2. Construire un Graphe d'Interférence Complet (GIC)
3. Sélectionner un sous-graphe maximisant la sécurité
4. Extraire les positions d'insertion de ce sous-graphe.

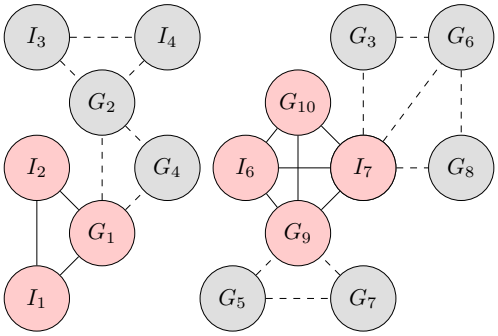
Example 4.2.1 (*Nouvelle approche de résolution du SLL*) : *Notre approche de résolution est illustrée dans la Figure 4.1. En partant du circuit de la Figure 4.1a, nous calculons le GIC de la Figure 4.1b. Chaque entrée et chaque porte est présente dans le GIC. Nous sélectionnons ensuite le sous-graphe maximisant la sécurité pour une valeur de $K = 7$ dans la Figure 4.1c avec un algorithme d'optimisation. Ce sous-graphe contient les positions d'insertions optimales pour la sécurité, nous obtenons alors le circuit verrouillé de la Figure 4.1d.*



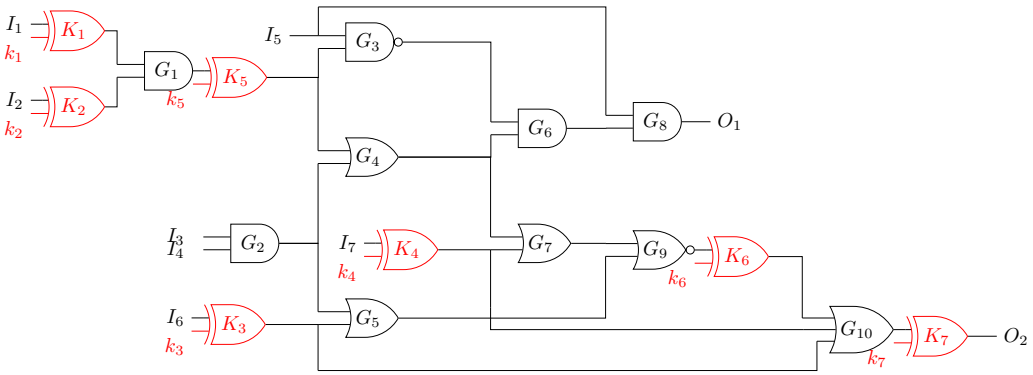
(a) Exemple de circuit combinatoire



(b) Graphe d'Interférence Complet du circuit Figure 4.1a



(c) Sous-graphe optimal pour $K = 7$ du GIC Figure 4.1b



(d) Circuit de la Figure 4.1a verrouillé avec une sécurité optimale pour 7 portes clés

Figure 4.1. : Nouvelle approche de sécurité pour un circuit

S	$NOT(S)$
0	1
1	0
X	-X
Y	-Y
-X	X
-Y	Y

S	$BUFFER(S)$
0	0
1	1
X	X
Y	Y
-X	-X
-Y	-Y

Tableau 4.1. : Tableaux de propagation de la porte NOT et BUFFER

Comme la plus grande clique, mesurant la sécurité, correspond à une partie irréductible de la clé à énumérer, nous proposons d'intégrer d'autres cliques disjointes à cette mesure. Elles correspondent également à des parties irréductibles de la clé. Nous mesurons alors la sécurité comme la somme des sécurités de chaque clique. Ainsi, notre mesure devient $\sum_{c \in C} 2^{|c|}$ avec C un ensemble de cliques disjointes. Comme C est composé d'éléments disjoints, il correspond à une partition d'un sous-graphe, appelée partition en cliques.

Example 4.2.2 (*Différence de mesure de la sécurité*) : *Dans la Figure 4.1b, en prenant $K = 7$, la plus grande clique est de taille 4 ($\{I_6, I_7, G_9, G_{10}\}$), cependant, ignorer la seconde clique correspond à ignorer une clique de taille 3 ($\{I_1, I_2, G_1\}$), soit 50% de la sécurité finale.*

Construction du GIC

Pour construire le GIC, nous proposons de calculer la relation d'interférence définie dans le *SLL*, en s'inspirant du *D-algorithm* [RBS67]. Cet algorithme cherche à trouver un vecteur d'entrée permettant de refléter l'état d'un signal noté D. Cet état est traité comme une inconnue dans le circuit, et une logique booléenne prenant en compte cette inconnue est créée. Dans notre cas, nous définissons deux inconnues X et Y ainsi que les tables de vérités de chaque portes logiques pour calculer ces inconnues. La Tableau 4.1 montre cette logique pour la porte NOT et $BUFFER$. En annexe A, les portes élémentaire AND, $NAND$, OR, NOR, XOR et $XNOR$ ont été définies dans les Tableau A.2, Tableau A.3 et Tableau A.4.

Ces portes ont été définies en se basant sur la définition orignelle d'interférence du *SLL* (Définition 3.4.2). L'objectif dans cette logique à deux inconnues est de propager autant que possible X tout en l'inhibant à chaque rencontre avec Y. Cette logique permet de déterminer si X est observable sur une sortie, malgré l'influence logique

de Y. Si, pour tous les vecteurs d'entrées I, X n'est pas observable, alors Y interfère avec X. En retour, si X interfère avec Y, alors les portes clés associées sont en relation *pairwise secure*.

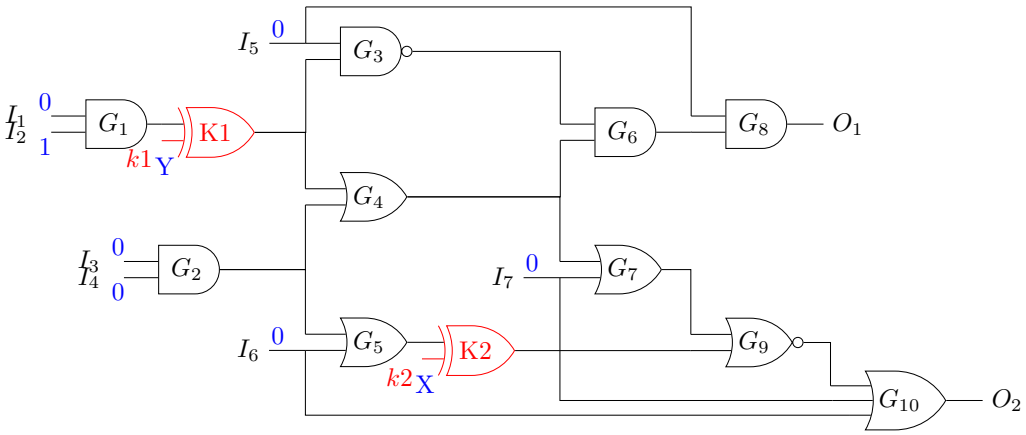
Example 4.2.3 (*Propagation d'un vecteur d'entrées*) : *La Figure 4.2 montre la propagation du vecteur* $(0,1,0,0,0,0,0)$ *pour l'interference de* G_1 *sur* G_5 *sur le circuit de la Figure 4.1a.*

La première étape (Figure 4.2a) insère le vecteur d'entrée, pose l'incoonue Y *pour le bit* k_1 *et* X *pour le bit* k_2.
La deuxième étape (Figure 4.2b) propage au travers de G_1, G_2 *et* K_1 *la valeur des signaux en fonction des tables prédeterminées (Tableau A.2 et Tableau A.4).*
La troisième étape (Figure 4.2c) propage au travers de G_3, G_4, G_5 *et* K_2 *la valeur des signaux en fonction des tables prédeterminées (Tableau A.2, Tableau A.3 et Tableau A.4).*
La quatrième étape (Figure 4.2d) propage au travers de G_6 *et* G_7 *la valeur des signaux en fonction des tables prédeterminées (Tableau A.2 et Tableau A.3).*
La cinquième étape (Figure 4.2e) propage au travers de G_8, G_9 *et* $G10$ *la valeur des signaux en fonction des tables prédeterminées (Tableau A.2 et Tableau A.3). Notons qu'un* $OU(a,b,c)$ *peut se réécrire comme un* $OU(OU(a,b),c)$, *on peut alors calculer la valeur de* $G10$ *par* $OU(OU(-Y,0),0) = OU(-Y,0) = -Y$.

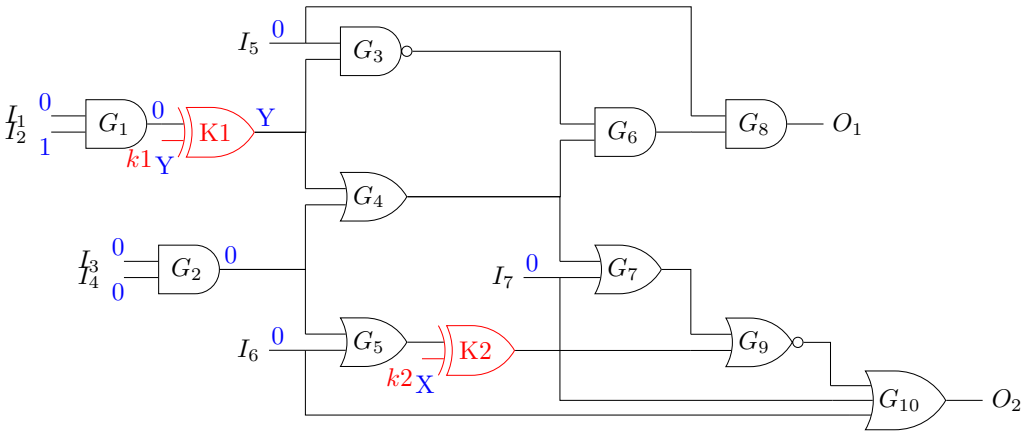
Comme il n'y a pas de X *dans une des sorties, le vecteur d'entrées* $(0,1,0,0,0,0,0)$ *ne permet pas d'observer* k_2 *dans une sortie indépendamment de* k_1.

Pour calculer les interférences, il faut calculer la propagation du circuit pour toutes les entrées. Cependant, en notant i le nombre d'entrées dans le circuit, il existe 2^i vecteurs d'entrées différents. Il est, en pratique, difficile de tester tous les vecteurs d'entrées pour un circuit avec un grand nombre d'entrées. Nous proposons donc de vérifier un sous-ensemble de vecteurs d'entrées, en choisissant des vecteurs générés par un algorithme de génération automatique de vecteurs de test (*ATPG*). L'objectif de ces vecteurs est de pouvoir détecter une faute lors de la fabrication du circuit (*stuck at 0 / 1 fault*). Cela permet d'avoir une exploration complète du circuit, même si, en pratique, les algorithmes cherchent à minimiser le nombre de vecteurs de tests.

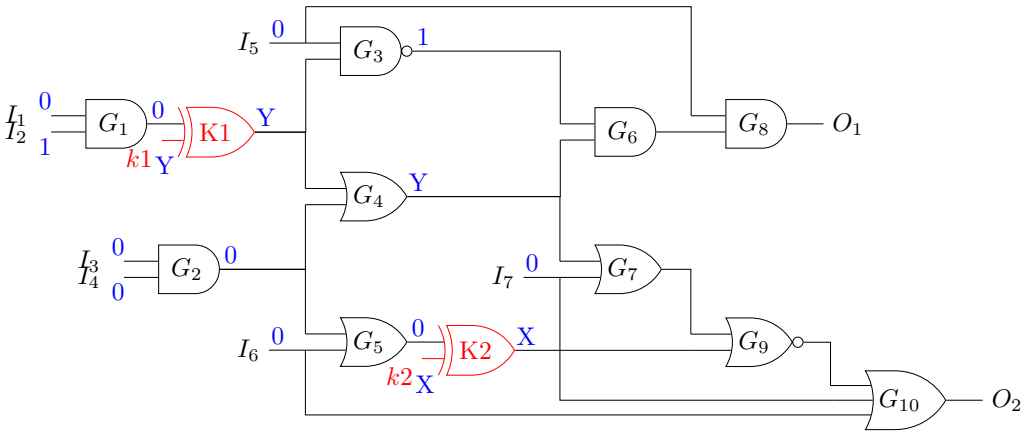
Si un circuit possède des portes logiques plus complexes à n entrées, nous pouvons créer les tables de vérité pour une logique à deux inconnues en considérant les portes comme un circuit. Nous pouvons alors calculer tous les vecteurs d'entrées possibles, ce qui fait 6^n vecteurs à tester. Si la porte complexe a un nombre d'entrées trop grand, il est possible de calculer seulement un sous-ensembles de couple entrée/sortie, à compléter au gré des rencontres.



(a) Étape 1 : ajout des valeurs du vecteur d'entrées et des inconues des bits clés

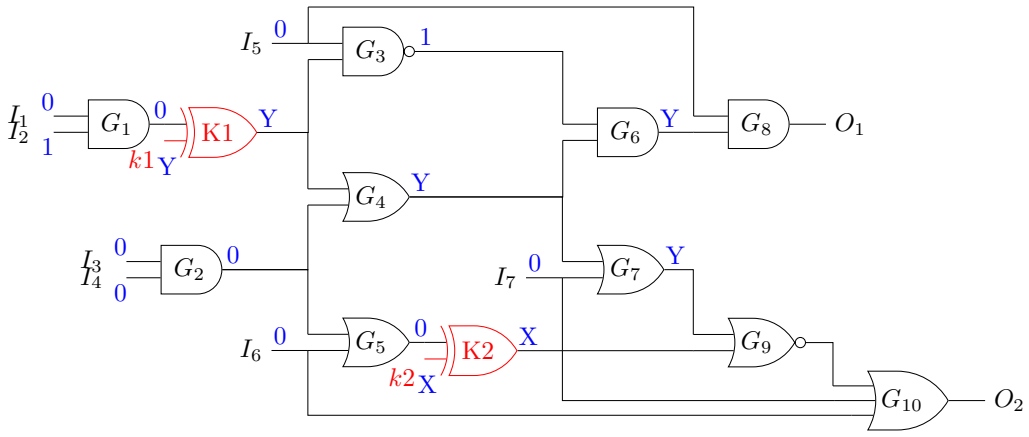


(b) Étape 2 : propagation des signaux au travers de G_1, G_2 et K_1

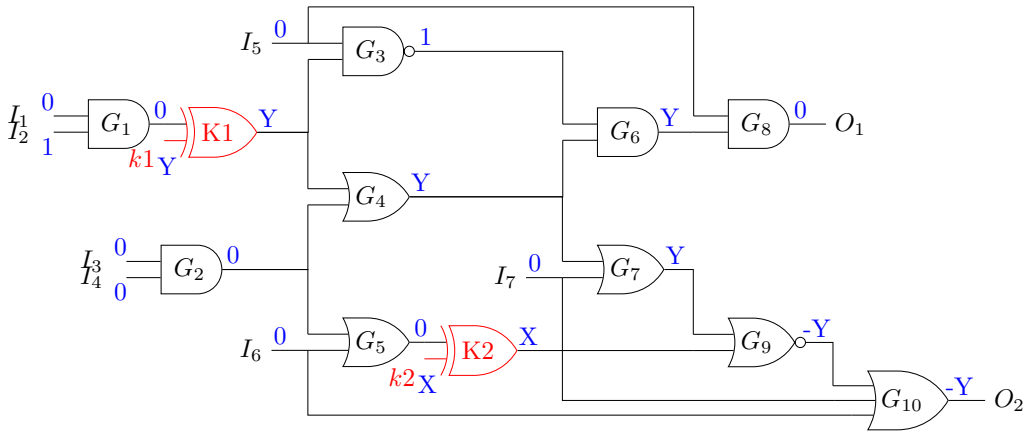


(c) Étape 3 : propagation des signaux au travers de G_3, G_4, G_5 et K_2

Figure 4.2. : Exemple de propagation du vecteur $(0, 1, 0, 0, 0, 0, 0)$ pour l'interference de G_1 sur G_5



(d) Étape 4 : propagation des signaux au travers de G_6 et G_7



(e) Étape 5 : propagation des signaux au travers de G_8, G_9 et G_{10}

Figure 4.2. : Exemple de propagation du vecteur $(0, 1, 0, 0, 0, 0, 0)$ pour l'interference de G_1 sur G_5 (suite)

4.3 Définition du problème de sécurité

Nous avons introduit une nouvelle mesure de sécurité. Elle est liée à une partition en cliques. Nous proposons un état de l'art de ce problème, puis une étude de complexité de notre problème.

Dans cette section, nous utiliserons un graphe $G = (V, E)$ avec V les sommets du graphe et E l'ensemble des arêtes. Notons $n = |V|$ et $m = |E|$ la taille de ces ensembles, et $\mathcal{P}(V)$ l'ensemble des parties de V.

4.3.1 Partition en cliques

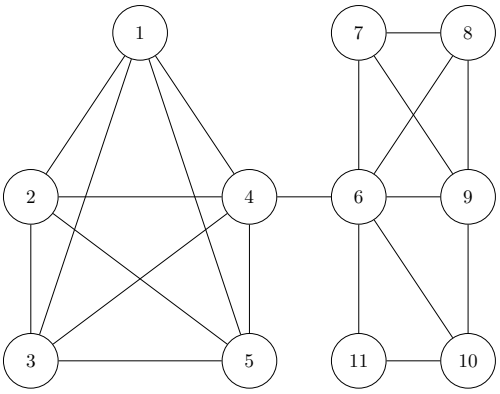
Le problème de partition en cliques (*Clique Cover : CC*) est un problème NP-complet [Kar75]. Il peut se modéliser comme suit :

$$CC = \begin{cases} \min & |P| & & (4.1) \\ \underline{s.t.} & V = \bigcup_{c \in P} c & & (4.2) \\ & c \cap c' = \emptyset & \forall c, c' \in P, c \neq c' & (4.3) \\ & (v, v') \in E & \forall c \in P,\ \forall v, v' \in c & (4.4) \\ & P \subseteq \mathcal{P}(V) & & (4.5) \end{cases}$$

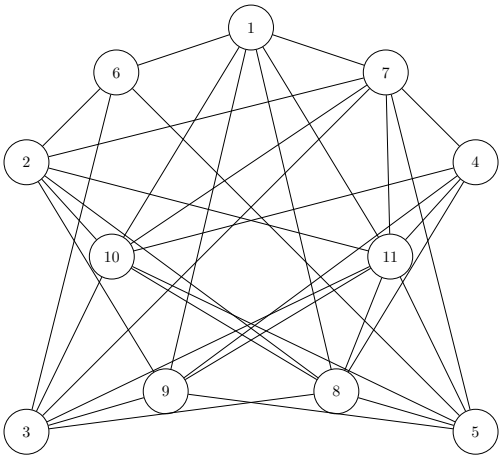
Soit $P \subseteq \mathcal{P}(V)$ un ensemble de parties de V, l'objectif (4.1) du problème est de minimiser le nombre de parties dans P. La contrainte (4.2) impose que chaque élément de V soit dans une partie de P. La contrainte (4.3) impose que chaque paire de parties soit disjointe. Ces deux contraintes imposent que P soit une partition. Enfin, la contrainte (4.4) impose que chaque partie de P forme une clique.

Le problème CC consiste à trouver la plus petite partition des sommets d'un graphe G, en minimisant le nombre de parties, sous la contrainte que chaque partie doit former une clique dans G. En prenant le graphe inverse $\overline{G}$, une partition en cliques de G devient une partition en stables de $\overline{G}$. Or, le problème de minimisation de partition en stables est plus communément appelé problème de coloration, dans lequel une couleur est attribuée à chaque sommet, avec la contrainte que deux sommets adjacents doivent avoir une couleur différente. Le problème de coloration cherche alors à minimiser le nombre de couleurs nécessaires pour colorer le graphe sous cette contrainte. La valeur minimale de ce problème est appelée le nombre chromatique du graphe. L'exemple dans la Figure 4.3 montre cette relation. Le

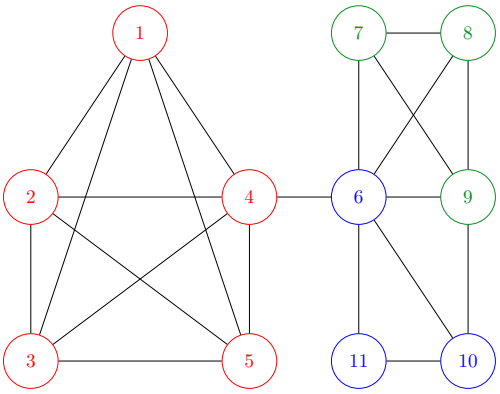
problème de coloration a été largement étudié dans la littérature [PMX98], et tous les résultats sur le problème de coloration sont transposables au problème de partition en cliques, comme la non-approximabilité du problème de coloration. Khanna et al. [KLS00] ont montré que sous l'hypothèse de $P \neq NP$, il n'existe pas d'algorithme polynomial permettant d'approcher le nombre chromatique d'un graphe G à n sommets, à n^{ϵ} près, $\forall \epsilon > 0$.



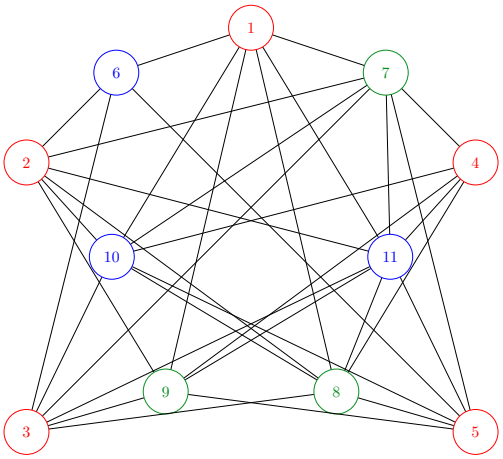
(a) Exemple d'un graphe



(b) Graphe inverse de Figure 4.3a



(c) Partition en cliques optimale du graphe Figure 4.3a



(d) Coloration optimale du graphe inverse Figure 4.3b

Figure 4.3. : Exemple de l'équivalence du problème de partition en cliques et du problème de coloration d'un graphe

4.3.2 Partition en cliques pondérées

Nous proposons de modéliser le problème de partition en cliques pondérées. Notons $w(c)$ le poids d'une clique c, dans notre cas $w(c) = 2^{|c|}$, et $w(P) = \sum_{c \in P} w(c)$ la somme des poids de chaque clique de la partition en cliques P.

Contrairement au problème CC, notre problème ne cherche pas à minimiser le nombre de parties, mais à maximiser la somme des poids de chaque clique. Appelons-le : problème de partitionnement en cliques pondérées (*Weighted Clique Covering : WCC*).

$$WCC = \begin{cases} \max & w(P) = \sum_{c \in P} w(c) & & (4.6) \\ \underline{s.t.} & V = \bigcup_{c \in P} c & & (4.7) \\ & c \cap c' = \emptyset & \forall c, c' \in P, c \neq c' & (4.8) \\ & (v, v') \in E & \forall c \in P,\ \forall v, v' \in c & (4.9) \\ & P \subseteq \mathcal{P}(V) & & (4.10) \end{cases}$$

Soit $P \subseteq \mathcal{P}(V)$ un ensemble de parties de V, l'objectif (4.6) du problème est de maximiser la somme des poids de chaque clique. Les contraintes (4.7) - (4.9) sont indentiques aux contraintes (4.2) - (4.4)

Entre le problème CC et WCC, seul l'objectif change. Maximiser le poids de chaque clique tend à maximiser le nombre d'éléments dans chaque partie et ainsi à minimiser le nombre de parties. Nous supposons alors que ce problème est également difficile à résoudre et nous le démontrons dans la section suivante. Néanmoins, les partitions optimales de chaque problème peuvent être différentes sur le même graphe.

Example 4.3.1 (*Exemple de solutions optimales différentes entre le problème CC et WCC*) : *Prenons le graphe de la Figure 4.4. La solution optimale du problème CC est :*

$$s^*_{CC} = \{\{1, 6, 7\}, \{2, 8, 9\}, \{3, 10, 11\}, \{4, 12, 13\}, \{5, 14, 15\}\}$$

Alors que la solution optimale du problème WCC est :

$$s^*_{WCC} = \{\{1, 2, 3, 4, 5\}, \{6, 7\}, \{8, 9\}, \{10, 11\}, \{12, 13\}, \{14, 15\}\}$$

Observons que $w(s^*_{WCC}) = 2^5 + 5 \times 2^2 = 52 > 40 = 5 \times 2^3 = w(s^*_{CC})$ et que $|s^*_{WCC}| = 6 > 5 = |s^*_{CC}|$. Les deux problèmes sont donc distincts.

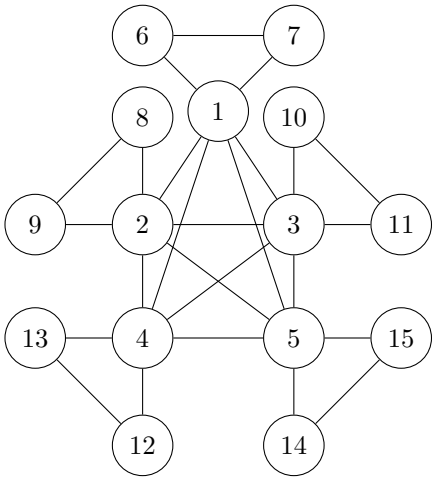


Figure 4.4. : Contre exemple pour différencier le problème de partitionnement en cliques et partitionnement en cliques pondérées

Le problème de sécurité est au moins aussi difficile que le problème WCC car nous recherchons un sous-graphe de taille K maximisant la valeur objective du problème WCC. Il est au moins aussi difficile car plus général, prenons l'exemple où $K = n$.

4.3.3 Complexité du problème WCC

Nous souhaitons nous intéresser à la complexité du problème. Pour ce faire, nous étudierons le cas particulier avec $K = n$, qui est le problème WCC. Nous allons prouver qu'il n'existe pas d'algorithme polynomial permettant d'obtenir une solution optimale. Cette preuve se base sur un résultat d'inapproximabilité du problème de la clique maximum *(Maximum Clique Problem : MCP)*, qui consiste à trouver la taille de la plus grande clique dans un graphe.

Pour cette preuve, nous utilisons les notations suivantes :

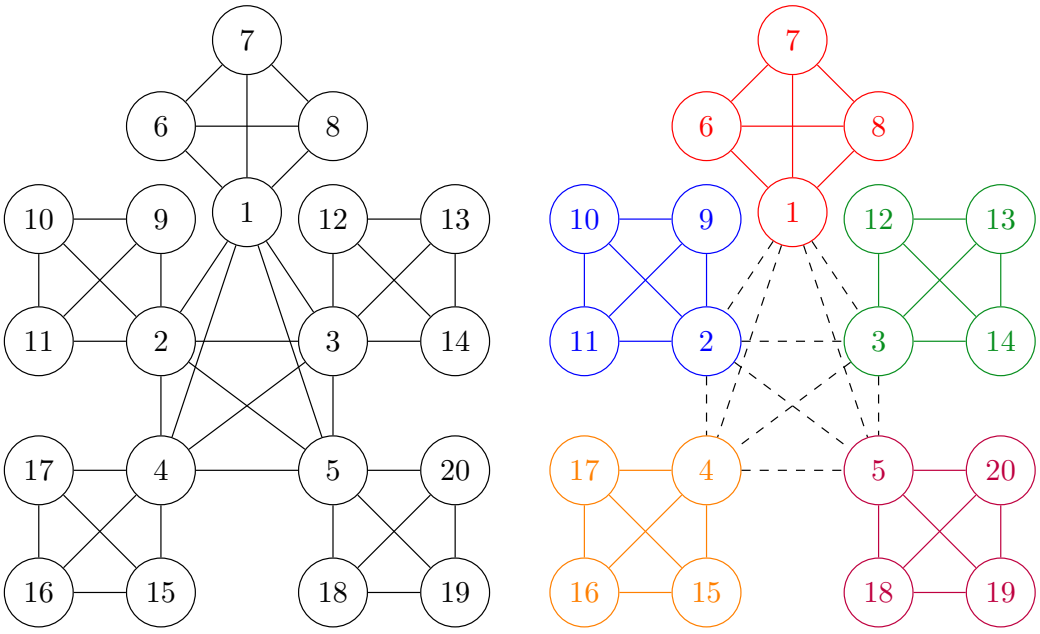
- $G = (V, E)$: un graphe avec $n = |V|$: la taille du graphe
- $z(P) = \sum_{c \in P} 2^{|c|}$: la valeur objective de la partition en cliques P du problème WCC.
- $\omega(G)$: la taille de la plus grande clique dans le graphe G
- $\omega(P)$: la taille de la plus grande clique dans la partition en cliques P
- $A^*(G)$: un algorithme de résolution exacte du problème WCC
- P^* : une partition en cliques optimale pour le problème WCC, retournée par $A^*(G)$.

Afin de démontrer que le problème WCC est $NP\text{-}DUR$, nous allons démontrer que :

1. La plus grande clique de la partition optimale P^*, résultat d'un algorithme exact $A^*(G)$ avec un ratio r constant du problème WCC, permet d'encadrer la taille de la plus grande clique du graphe.
2. Cet encadrement ne peut pas être obtenu avec un algorithme polynomial, donc l'algorithme $A^*(G)$ n'est pas polynomial.

Pour commencer, montrons que la solution optimale ne contient pas toujours la plus grande clique.

Example 4.3.2 (*Exemple de partition optimale à WCC qui ne contient pas la plus grande clique*) : *Prenons le contre-exemple de la Figure 4.5. La solution optimale a une valeur objective de $5 \times 2^4 = 80$, alors que la partition contenant la plus grande clique et maximisant l'objectif a une valeur objective $2^5 + 5 \times 2^3 = 72$.*



(a) Graphe contre-exemple **(b)** Partition en cliques pondérées optimale

Figure 4.5. : Contre exemple montrant que la solution optimale du WCC ne contient pas la plus grande clique

Inapproximabilité de MCP

Premièrement, voici le résultat d'inapproximabilité de MCP.

Théorème 4.3.1. *Si $P \neq NP$, pour tout $\epsilon > 0$, il n'existe pas d'algorithme en temps polynomial qui approche la taille de la plus grande clique $\omega(G)$ dans un graphe $G = (V, E)$ avec un ratio inférieur à $n^{1-\epsilon}$.*

Démonstration. Zuckerman [Zuc06] le prouve sous l'hypothèse $P \neq NP$. □

Lemmes

Notre preuve de $NP\text{-}DUR$ nécessite trois lemmes intermédiaires, présentés ci-dessous, les preuves sont en Annexe B.

Lemme 4.3.2. $\forall x \in \mathbb{R}^+, x \geq 2, \forall a \in \mathbb{R}^+, a \geq 2, x + a \leq xa.$

Lemme 4.3.3. $\forall x \in \mathbb{R}^+, x \geq 16$, $log_2(x) \leq \sqrt{x}$

Lemme 4.3.4. *Pour une partition en cliques donnée P et un entier fixé $\Delta \in \mathbb{N}$ tel que $1 \leq \Delta < \omega(P)$,*

$$\underset{k \in \{0,\ldots,\Delta\}}{\operatorname{argmin}} \left\{ 2^{\Delta+k} \times (\omega(P) - k) \mid \omega(P) > \Delta \geq k \geq 0 \right\} = 0$$

Encadrement

En utilisant les lemmes précédents, nous allons prouver le résultat d'encadrement.

Théorème 4.3.5. *Soit $G = (V, E)$ un graphe, et soit $n = |V|$. Soit P^* la partition en cliques résultat de $A^*(G)$. Soit $\omega(G)$ la taille de la plus grande clique dans G, et $\omega(P^*)$ la taille de la plus grande clique dans la partition optimale P^*. L'inégalité suivante est vérifiée : $\omega(P^*) \leq \omega(G) \leq \omega(P^*) + \log_2(n)$.*

Démonstration. Soit $\omega(G)$ la taille de la plus grande clique dans le graphe G. Soit $z(P) = \sum_{c \in P} 2^{|c|}$ la valeur de la fonction objectif de la solution P. Soit P^* une partition en cliques optimale pour WCC. Soit $\overline{P}$ la meilleure solution contenant une clique de taille $\omega(G)$. Par définition de l'optimalité, nous avons $z(\overline{P}) \leq z(P^*)$. Remarquons que $\omega(P^*) \leq \omega(G) \iff \exists \Delta \in \mathbb{N},\ \omega(P^*) + \Delta = \omega(G)$. Pour compenser la différence de valeur objectif entre $z(P^*)$ et $z(\overline{P})$, nous devons compenser avec $2^{\omega(G)} = 2^{\Delta + \omega(P)} = 2^{\Delta} \times 2^{\omega(P)}$. Cela nécessite au moins $2^{\Delta} \times \omega(P^*)$ sommets dans 2^{Δ} cliques disjointes de taille $\omega(P^*)$. Selon le Lemme 4.3.4, compenser avec des cliques de taille plus petite nécessiterait plus de sommets. Cependant, nous avons un nombre limité de sommets n. Ainsi, nous obtenons

$$
\begin{aligned}
& 2^{\Delta} \times \omega(P^*) \leq n \iff 2^{\Delta} \leq \frac{n}{\omega(P^*)} && \text{puisque } \omega(P^*) \geq 1 \\
\iff & \log_2(2^{\Delta}) \leq \log_2\left(\frac{n}{\omega(P^*)}\right) \\
\iff & \Delta \leq \log_2(n) - \log_2(\omega(P^*)) \\
\iff & \omega(G) - \omega(P^*) \leq \log_2(n) - \log_2(\omega(P^*)) \\
\iff & \omega(G) - \log_2(n) \leq \omega(P^*) - \log_2(\omega(P^*)) \\
\implies & \omega(G) - \log_2(n) \leq \omega(P^*)
\end{aligned}
$$

Cependant, puisque $\omega(P^*) \leq \omega(G)$ par définition, nous obtenons :

$$
\omega(G) - \log_2(n) \leq \omega(P^*) \leq \omega(G) \iff \omega(P^*) \leq \omega(G) \leq \omega(P^*) + \log_2(n) \quad (4.11)
$$

Par conséquent, la solution optimale de WCC fournit une approximation de $\omega(G)$ à un écart de $\log_2(n)$ près. □

NP-DUR

Nous avons tous les éléments pour prouver que le problème WCC est NP-DUR.

Théorème 4.3.6. *Soit $A^*(G)$ un algorithme de résolution exacte du problème WCC. Pour tous les graphes $G = (V, E)$ tels que $|V| \geq 16$, $A^*(G)$ n'est pas polynomial. Le problème WCC est alors NP-DUR.*

Démonstration. Soit $G = (V, E)$ un graphe, et soit $n = |V|$. Soit $A^*(G)$ un algorithme de résolution exacte du problème WCC. Soit P^* la partition en cliques optimale résultat de $A^*(G)$. Soit $\omega(G)$ la taille de la plus grande clique dans G et $\omega(P^*)$ la

taille de la plus grande clique dans P^*. Selon le Théorème 4.3.5, $A^*(G)$ permet de trouver une borne de $\omega(G)$:

$$\begin{aligned}
&\omega(P^*) \leq \omega(G) \leq \omega(P^*) + \log_2(n) \\
\iff &\omega(P^*) \leq \omega(G) \leq \omega(P^*) \times \log_2(n) && \text{(Pour } n \geq 4\text{, c.f. Lemme 4.3.2)} \\
\iff &\omega(P^*) \leq \omega(G) \leq \omega(P^*) \times \sqrt{n} = \omega(P^*) \times n^{\frac{1}{2}} && \text{(Pour } n \geq 16\text{, c.f. Lemme 4.3.3)}
\end{aligned}$$

Cependant, selon le Théorème 4.3.1, il n'existe pas d'algorithme polynomial pour approximer $\omega(G)$ avec un ratio inférieur à $n^{1-\epsilon}$ pour tout $\epsilon > 0$, en particulier pour $\epsilon = \frac{1}{2}$. Par conséquent, $A^*(G)$ n'est pas polynomial, il n'existe donc pas d'algorithme polynomial pour trouver la solution optimale de WCC. Ainsi, le problème WCC est NP-DUR. □

Grâce au Théorème 4.3.6, nous avons prouvé que le problème WCC est $NP\text{-}DUR$. Le problème WCC étant un cas particulier de notre problème de sécurité avec $K = n$, il n'existe pas d'algorithme de résolution polynomial pour notre problème.

4.4 Modélisation du problème de sécurité

Ce problème de sécurité est $NP\text{-}DUR$, mais nous souhaitons malgré tout le résoudre, au moins pour des tailles d'instances raisonnables. Nous envisageons alors de modéliser le problème et d'observer le comportement des algorithmes de résolution sur nos cas d'application. Nous proposons deux modélisations du problème : une première directe, puis une seconde nécessitant un prétraitement.

4.4.1 Modélisation d'emplacements de cliques dynamiques

Dans ce modèle, nous devons sélectionner un sous-graphe de taille K, définir une partition en cliques de ce sous-graphe et calculer la somme des poids des cliques. Pour ce faire, nous allons créer un modèle qui a K emplacements de groupes de sommets, représentant des cliques dynamiques. Les sommets du sous-graphe seront assignés à ces emplacements, en contraignant qu'une clique soit formée dans ces emplacements.

Les données sont :

— $G = (V, E)$: le graphe d'interférence complet du circuit (Section 4.2) de G

— K : la taille du sous-graphe, i.e. le nombre de portes logiques à ajouter dans le circuit

— $\sigma \in \Sigma = \{1, \ldots, K\}$: les emplacements de cliques dans le modèle

Soit la variable de décision : $x_v = \begin{cases} 1 & \text{si le sommet } v \text{ est sélectionné dans le sous-graphe} \\ 0 & \text{sinon} \end{cases}$

Notons les variables de modélisation :

— $w_\sigma \in \{0, \ldots, K\}$: la taille de la clique dans l'emplacement σ

— $\lambda_\sigma = \begin{cases} 1 & \text{si l'emplacement } \sigma \text{ est utilisé} \\ 0 & \text{sinon} \end{cases}$

— $a_\sigma^v = \begin{cases} 1 & \text{si le sommet } v \text{ est positionné dans l'emplacement } \sigma \\ 0 & \text{sinon} \end{cases}$

L'objectif (4.12) est de maximiser la somme des poids des cliques de la partition. La multiplication avec λ_σ permet de ne pas compter les cliques vides.

$$\max \sum_{\sigma \in \Sigma} 2^{w_\sigma} * \lambda_\sigma \tag{4.12}$$

La contrainte de sac à dos (4.13) impose un nombre maximum de sommets K dans le sous-graphe sélectionné. Ce qui correspond au nombre de portes clés à ajouter dans le circuit.

$$\sum_{v \in V} x_v \leq K \tag{4.13}$$

La contrainte (4.14) compte le nombre d'éléments dans l'emplacement de clique.

$$w_\sigma = \sum_{v \in V} a_\sigma^v \qquad \forall \sigma \in \Sigma \tag{4.14}$$

Les contraintes de sélection (4.15) et (4.16) imposent qu'un emplacement de clique σ soit utilisé si et seulement s'il y a au moins un élément assigné à cet emplacement σ.

$$\lambda_\sigma \leq w_\sigma \qquad \forall \sigma \in \Sigma \tag{4.15}$$

$$\lambda_\sigma \geq a_\sigma^v \qquad \forall \sigma \in \Sigma,\ \forall v \in V \tag{4.16}$$

La contrainte de disjonction de cliques (4.17) impose qu'un sommet v du sous-graphe solution soit assigné à au plus un emplacement σ, et qu'un sommet hors du sous-graphe ne soit pas assigné.

$$\sum_{\sigma \in \Sigma} a_\sigma^v \leq x_v \qquad \forall v \in V \tag{4.17}$$

La contrainte d'intégrité de clique (4.18) impose que deux éléments v, v' présents dans un même emplacement σ soient voisins dans le graphe d'interférence E, une valeur représentée par $M[v,v'] = \begin{cases} 1 & \text{si } \{v,v'\} \in E \\ 0 & \text{sinon} \end{cases}$

$$a_\sigma^v + a_\sigma^{v'} \leq 1 + M[v,v'] \qquad \forall (v,v') \in V^2,\ v \neq v',\ \forall \sigma \in \Sigma \tag{4.18}$$

$$M_1 = \begin{cases} \max & \displaystyle\sum_{\sigma \in \Sigma} 2^{w_\sigma} * \lambda_\sigma & & (4.11) \\ \underline{s.t.} & \displaystyle\sum_{v \in V} x_v \leq K & & (4.12) \\ & w_\sigma = \displaystyle\sum_{v \in V} a_\sigma^v & \forall \sigma \in \Sigma & (4.13) \\ & \lambda_\sigma \leq w_\sigma & \forall \sigma \in \Sigma & (4.14) \\ & \lambda_\sigma \geq a_\sigma^v & \forall \sigma \in \Sigma,\ \forall v \in V & (4.15) \\ & \displaystyle\sum_{\sigma \in \Sigma} a_\sigma^v \leq x_v & \forall v \in V & (4.16) \\ & a_\sigma^v + a_\sigma^{v'} \leq 1 + M[v,v'] & \forall (v,v') \in V^2,\ v \neq v',\ \forall \sigma \in \Sigma & (4.17) \\ & 0 \leq w_\sigma \leq K, \quad \lambda_\sigma \in \mathbb{B} & \forall \sigma \in \Sigma & (4.18) \\ & x_v \in \mathbb{B}\ \forall v \in V \qquad a_\sigma^v \in \mathbb{B} & \forall v \in V,\ \forall \sigma \in \Sigma & (4.19) \end{cases}$$

Le modèle M_1 représente ce problème, il contient $\mathcal{O}(K*|V|)$ variables et $\mathcal{O}(K*|V|^2)$ contraintes. Nous avons résolu le modèle M_1 avec $Gurobi$ [Gur23]. Les algorithmes ont été implémentés en Julia 1.8 [Bez+17], et les calculs ont été effectués sur un Windows 10 64 bits, Intel $i7-8665U$ avec 16 Go de mémoire. Le Tableau 4.2 présente les informations des circuits *ISCAS-85* [BF85], les informations du modèle et les résultats obtenus. Le $TimeOut$ est fixé à 60 minutes. Le nombre de portes clés est un pourcentage du nombre de portes logiques du circuit. Constatons que la création du modèle est difficile pour des circuits à 1000 sommets et la résolution pour des circuits à 500 sommets. Lors de nos résolutions, nous avons constaté que les modèles

devenaient incohérents lorsque la solution optimale dépassait un seuil situé entre 2^{49} et 2^{74}. Nous estimons que ce problème provient de la limite de représentation mémoire d'un nombre dans les solveurs. Par exemple, pour le circuit $c3540$ avec un $K = 74(15\%)$, le résultat est une clique de taille 74 mais la valeur objective retourné par le solveur est 0.

Information du circuit				Modèle M_1			
circuit	#sommets	#portes clés	sécurité optimale	#variables	#contraintes	Création modèle (s)	Temps optimisation (s)
c499	207	10 (5%)	$2^{10.0}$	2417	358978	1.21	2.85
		(10%)	$2^{21.0}$	5079	754549	3.63	7.09
		(15%)	$2^{31.0}$	7709	1114999	4.9	18.1
c880	228	11 (5%)	$2^{7.1699}$	2890	566135	2.24	6.44
		23 (10%)	$2^{8.5999}$	5794	1183487	5.72	19.43
		34 (15%)	$2^{9.1699}$	8456	1749393	8.73	27.0
c1355	207	10 (5%)	$2^{10.0}$	2417	367898	1.46	3.59
		21 (10%)	$2^{21.0}$	5079	773281	3.13	7.79
		31 (15%)	$2^{31.0}$	7709	1142651	6.11	25.09
c3540	491	25 (5%)	$2^{25.0}$	13491	5735417	33.64	86.88
		49 (10%)	$2^{49.0}$	27147	11245649	76.48	218.83
		74 (15%)	X	-	-	-	-
c5315	758	38 (5%)	X	31158	21577615	151.57	TimeOut
		76 (10%)	X	64446	43166023	1766	TimeOut
		114 (15%)	X	X	X	TimeOut	X
c6288	1480	74 (5%)	X	X	X	TimeOut	X
		148 (10%)	X	X	X	TimeOut	X
		222 (15%)	X	X	X	TimeOut	X
c7552	949	47 (5%)	X	X	X	TimeOut	X
		95 (10%)	X	X	X	TimeOut	X
		142 (15%)	X	X	X	TimeOut	X

Tableau 4.2. : Résultat du modèle M_1 sur les circuit $ISCAS - 85$

Example 4.4.1 (*Symétrie du modèle M_1*) : *Prenons le GIC de la Figure 4.1b et $K = 7$. La solution optimale du problème est un sous-graphe partitionné en deux cliques (Figure 4.1c) : $\{\{I_1, I_2, G_1\}; \{I_6, I_7, G_9, G_{10}\}\}$. Une solution optimale du modèle M_1 peut être :*

- *Emplacement 1 : $\{I_1, I_2, G_1\}$*
- *Emplacement 2 : $\{I_6, I_7, G_9, G_{10}\}$*

ou

- *Emplacement 1 : $\{I_6, I_7, G_9, G_{10}\}$*
- *Emplacement 2 : $\{I_1, I_2, G_1\}$*

Les deux solutions sont équivalentes, mais du point de vue mathématique, ces deux solutions sont différentes. Elles forment ce qu'on appelle une symétrie et peuvent ralentir la résolution.

Nous remarquons que les emplacements engendrent beaucoup de symétries dans le modèle. Nous supposons qu'elles ralentissent la résolution. Comme nous avons le GIC, nous pouvons pré-calculer toutes les cliques, puis créer un modèle qui n'a pas besoin de les calculer à la résolution, réduisant ainsi toutes les symétries liées aux emplacements.

4.4.2 Modélisation par liste de cliques maximales

Une difficulté du problème est de sélectionner un sous-graphe en formant des cliques. La résolution du problème serait plus simple si nous avions toutes les cliques du graphe d'interférence. Cependant, il y a un nombre exponentiel de cliques maximales dans un graphe : $\mathcal{O}(3^{\frac{n}{3}})$ [MM65]. Dans nos circuits, la densité des graphes d'interférence est en moyenne de 0.0417. Cela indique que seulement 4.17% des arêtes du graphe sont présentes. Les graphes ayant une faible densité d'arêtes sont appelés **graphe creux** (*sparse graph*). Des algorithmes efficaces pour la génération de Listes de Cliques Maximales (LCM) ont été proposés pour des graphes creux [ELS10]. Utiliser un algorithme exponentiel en pré-traitement d'un problème NP-DUR, semble contre-intuitif, néanmoins, si $P \neq NP$, un algorithme de résolution exacte de problème NP-DUR est exponentiel. Nous observerons qu'en pratique, cela améliore les temps de calculs.

En modélisant chaque clique de la LCM comme un emplacement où seuls les éléments de la clique peuvent être associés, on peut modifier le modèle précédent pour utiliser la représentation LCM du graphe. Ainsi, nous obtenons un modèle avec chaque clique qui est un sous-ensemble d'une clique maximale, ce qui forme une partition en cliques. Remarquons qu'il existe une unique liste de cliques maximales par graphe et un unique graphe associé à une telle liste. Cette liste est donc une représentation complète d'un graphe. Les données sont :

- $G = (V, E)$: le graphe d'interférence complet du circuit (Section 4.2) de G
- K : la taille du sous-graphe, i.e. le nombre de portes logiques à ajouter dans le circuit
- L : la liste de cliques maximales du graphe G
- $c \in L = \{c_1, \ldots, c_{|L|}\}$: les emplacements de cliques dans le modèle

Soit la variable de décision : $x_v = \begin{cases} 1 & \text{si le sommet } v \text{ est sélectionné} \\ & \text{dans le sous-graphe} \\ 0 & \text{sinon} \end{cases}$

Notons les variables de modélisation :

— $w_c \in \{0, \ldots, W_c\}$: la taille de la clique sélectionnée dans la clique maximale c, avec $W_c = |c|$, la taille de la clique maximale c.

— $\lambda_c = \begin{cases} 1 & \text{si la clique maximale } c \text{ est utilisée} \\ 0 & \text{sinon} \end{cases}$

— $a_c^v = \begin{cases} 1 & \text{si le sommet } v \text{ est positionné dans la clique } c \\ 0 & \text{sinon} \end{cases}$

L'objectif (4.20), la contrainte de sac à dos (4.21), la contrainte de taille (4.22), les contraintes de sélections (4.23) et (4.24), ainsi que la contrainte de disjonction de cliques (4.25) sont identiques au modèle M_1, au renommage de σ en c et aux nombres de contraintes pour chaque équation.

La contrainte d'intégrité de clique (4.26) est modifiée, il suffit de contraindre que l'on ne peut pas associer de sommets autres que ceux déjà présents dans la clique maximale c.

Nous obtenons le modèle M_2 qui contient $\mathcal{O}(|L| * |V|)$ variables et $\mathcal{O}(|L| * |V|)$ contraintes. Comme la taille de la liste L est exponentielle dans le pire cas, ce modèle a théoriquement un nombre exponentiel de contraintes et de variables.

$$
M_2 = \begin{cases}
\max & \displaystyle\sum_{c \in L} 2^{w_c} * \lambda_c & & (4.20) \\
\underline{s.t.} & \displaystyle\sum_{v \in V} x_v \leq K & & (4.21) \\
& w_c = \displaystyle\sum_{v \in C} a_c^v & \forall c \in L & (4.22) \\
& \lambda_c \leq w_c & \forall v \in V,\ \forall c \in L & (4.23) \\
& \lambda_c \geq a_c^v & \forall v \in V,\ \forall c \in L & (4.24) \\
& \displaystyle\sum_{c \in L} a_c^v \leq x_v & \forall v \in V & (4.25) \\
& a_c^v = 0 & \forall c \in L,\ \forall v \notin c & (4.26) \\
& x_v \in \{0, 1\} & \forall v \in V & (4.27) \\
& \lambda_c \in \{0, 1\},\ w_c \in \{0, \ldots, W_c\} & \forall c \in L & (4.28) \\
& a_c^v \in \{0, 1\} & \forall c \in L,\ \forall v \in V & (4.29)
\end{cases}
$$

Le Tableau 4.3 présente les résultats du modèle M_1 et modèle M_2 obtenus avec *Gurobi* [Gur23], dans les mêmes conditions que précédement. Ce tableau contient les informations des circuits, ainsi que les informations des modèles, leurs temps

Information du circuit				Modèle M_1				Modèle M_2			
circuit	#sommets	#portes clés	sécurité optimale	#variables	#contraintes	Création modèle (s)	Temps optimisation (s)	#variables	#contraintes	Création modèle (s)	Temps optimisation (s)
c499	207	10 (5%)	$2^{10.0}$	2417	358978	1.21	2.85	1746212	4201557	84.84	65.54
		(10%)	$2^{21.0}$	5079	754549	3.63	7.09	1746212	4201557	73.73	106.82
		(15%)	$2^{31.0}$	7709	1114999	4.9	18.1	1746212	3383109	65.17	88.46
c880	228	11 (5%)	$2^{7.1699}$	2890	566135	2.24	6.44	1437	2386	0.07	0.11
		23 (10%)	$2^{8.5999}$	5794	1183487	5.72	19.43	1437	2386	0.02	0.06
		34 (15%)	$2^{9.1699}$	8456	1749393	8.73	27.0	1437	2070	0.03	0.11
c1355	207	10 (5%)	$2^{10.0}$	2417	367898	1.46	3.59	553160	1323879	32.47	36.03
		21 (10%)	$2^{21.0}$	5079	773281	3.13	7.79	553160	1323879	46.47	195.46
		31 (15%)	$2^{31.0}$	7709	1142651	6.11	25.09	553160	1066973	37.97	20.38
c3540	491	25 (5%)	$2^{25.0}$	13491	5735417	33.64	86.88	3211	4577	0.09	0.07
		49 (10%)	$2^{49.0}$	27147	11245649	76.48	218.83	3235	4673	0.1	0.09
		74 (15%)	X	-	-	-	-	-	-	-	-
c5315	758	38 (5%)	$2^{38.0}$	31158	21577615	151.57	TimeOut	5215	7600	0.14	0.13
		76 (10%)	X	64446	43166023	1766	TimeOut	-	-	-	-
		114 (15%)	X	X	X	TimeOut	X	-	-	-	-
c6288	1480	74 (5%)	$2^{27.0000}$	X	X	TimeOut	X	10373	15270	0.37	0.34
		148 (10%)	$2^{27.0000}$	X	X	TimeOut	X	10373	15270	0.29	0.23
		222 (15%)	$2^{27.0000}$	X	X	TimeOut	X	10373	15270	0.28	0.27
c7552	949	47 (5%)	$2^{47.0}$	X	X	TimeOut	X	5887	8411	0.82	0.18
		95 (10%)	X	X	X	TimeOut	X	-	-	-	-
		142 (15%)	X	X	X	TimeOut	X	-	-	-	-

Tableau 4.3. : Comparaison du modèle M_1 et M_2 sur les circuit $ISCAS-85$

de création et leurs temps de résolution. Le $TimeOut$ est fixé à 60 minutes. Le nombre de portes clés est un pourcentage du nombre de portes logiques du circuit. Nous constatons que la création du modèle M_2 est plus lente que la création du modèle M_1 pour les petits circuits. Il est à noter que le temps de calcul des listes de cliques maximales est inclus dans la création du modèle M_2. Ce temps est corrélé au grand nombre de variables et de contraintes du modèle. Cependant, la création du modèle M_1 échoue pour les grands circuits, avec une durée limite (*TimeOut*) fixée à 60 minutes pour la création du modèle. Pour le circuit $c5315$, le modèle M_1 est créé mais sa résolution n'a pas abouti dans les 10 minutes imparties. Le modèle M_2 permet de résoudre des circuits de plus grandes tailles, comme le circuit $c6288$. Nous obtenons la même incohérence que précédement lorsque les solutions optimales dépassent un seuil (entre 2^{49} et 2^{74}).

Le modèle M_2 résout donc plus d'instance que le modèle modèle M_1, nous voulons proposer un algorithme dédié pour ce modèle afin de résoudre plus d'instances. Nous souhaitons également prendre en charge la limitation de l'objectif. Nous proposons alors un algorithme *Branch and Bound* pour résoudre le modèle M_2, présenté dans la section suivante.

4.5 Algorithme de résolution dédié

Nous proposons un algorithme *Branch and Bound* permettant de calculer efficacement les solutions optimales de notre problème, en particulier le modèle M_2.

Premièrement, nous allons définir un nouvel objectif, qui évite les problèmes de capacité mémoire. Ensuite nous allons proposer un calcul de borne supérieure. Cette étape est importante, car le calcul de borne permet de couper l'arbre de recherche afin de ne pas explorer tout l'espace de recherche. Nous allons ensuite proposer notre algorithme de *Branch and Bound*. Nous proposons également une stratégie de résolution itérative permettant de réduire les temps de calcul pour les algorithmes de résolution du modèle M_2 Enfin nous allons comparer les temps d'éxécution du solveur $Gurobi$ [Gur23] avec l'algorithme *Branch and Bound*.

4.5.1 Nouvel objectif

L'objectif des modèles M_1 et M_2 est exponentiel, et nous avons observé que dans certains cas, une trop grande valeur objective entraîne des incohérences dans la résolution. Nous proposons donc de prendre le logarithme de la fonction objective. Le logartihme ne modifie pas les solutions optimales car la fonction est strictement croissante.

Notons $P = \{c_1, \ldots, c_p\}$ une partition en cliques.

$$z(P) = \sum_{c \in P} 2^{|c|} \iff \log_2\left(z(P)\right) = \log_2\left(\sum_{c \in P} 2^{|c|}\right) \tag{4.30}$$

$$\iff \log_2\left(z(P)\right) = \log_2\left(2^{|c_1|} * \sum_{c \in P} \frac{2^{|c|}}{2^{|c_1|}}\right) \tag{4.31}$$

$$\iff \log_2\left(z(P)\right) = |c_1| + \log_2\left(\frac{2^{|c_1|}}{2^{|c_1|}} + \sum_{c \in P \setminus c_1} 2^{|c|-|c_1|}\right) \tag{4.32}$$

$$\iff \log_2\left(z(P)\right) = |c_1| + \log_2\left(1 + \sum_{c \in P \setminus c_1} 2^{|c|-|c_1|}\right) \tag{4.33}$$

En s'assurant que $|c_1| \geq |c_i| \forall i \in \{2, \ldots, p\}$, nous avons l'assurance que $0 < 2^{|c_i|-|c_1|} \leq 1$. Or une partition possède au plus K parties (des singletons) :

$$\log_2(z(P)) = |c_1| + \log_2\left(1 + \sum_{c \in P \setminus c_1} 2^{|c|-|c_1|}\right) \tag{4.34}$$

$$\implies \log_2(z(P)) \leq |c_1| + \log_2\left(1 + \sum_{c \in P \setminus c_1} 1\right) \tag{4.35}$$

$$\implies \log_2(z(P)) \leq |c_1| + \log_2(1 + K - 1) \tag{4.36}$$

$$\implies \log_2(z(P)) \leq |c_1| + \log_2(K) \tag{4.37}$$

Tant que $|c_1|$ et K ne dépassent pas les capacités mémoires, la fonction objective retournera un résultat cohérant. L'Algorithme 2 retourne la valeur de notre nouvel objectif, qui est le logarithme de l'objectif initial. Nous pouvons maintenant définir une borne supérieure pour cet objectif.

Algorithme 2 : $objective(partition)$

Entrées : partition : partition en cliques
Résultat : Retourne le logarithme de la valeur objective de la partition en cliques pondérées.

1 $c_1 \leftarrow \underset{c \in partition}{\text{argmax}} |c|$
2 $somme \leftarrow 1$
3 **pour tous les** $clique \in partition \setminus c_1$ **faire**
4 $\quad somme \mathrel{+}= 2^{|clique|-|c_1|}$
5 **fin**
6 **return** $c_1 + log_2(somme)$

4.5.2 Calcul de borne supérieure

Sans la contrainte de disjonction, la solution optimale est composée des plus grandes cliques, jusqu'à satisfaire la contrainte de sac à dos. L'algorithme est donc polynomial.

L'algorithme 3 calcule la borne supérieure d'une solution courante en ignorant cette contrainte pour la fin de la liste. La variable $index$ sert de pointeur dans la liste de cliques maximales, pour ne pas réutiliser des cliques utilisées et/ou non sélectionnées. De la ligne 1 à la ligne 5, l'algorithme calcule la valeur objective de la partition courante. Ensuite la boucle à la ligne 9 permet de calculer le cardinal

Algorithme 3 : $upperBoundM_2(L, current, K, index)$

Entrées : L : Liste des cliques maximales
current : Liste de cliques disjointes, formant la solution courante
K : Taille limite du sous-graphe
index : Entier indiquant la position courante dans la liste L
Résultat : Retourne la borne supérieure pour la solution courante.

1 $c_1 \leftarrow \underset{c \in current}{\text{argmax}} |c|$
2 $somme \leftarrow 1$
3 **pour tous les** $clique \in current \backslash c_1$ **faire**
4 $\quad somme \mathrel{+}= 2^{|clique|-|c_1|}$
5 **fin**
6 $cards \leftarrow Int[]$;
7 $vertices \leftarrow \bigcup_{c \in current} c$
8 $nbElem = card(vertices)$
9 **pour tous les** $clique \in L[index : end]$ **faire**
10 $\quad ajouter(cards, card(clique \backslash vertices))$
11 **fin**
12 $sort(cards)$; $iter = 0$
13 **tant que** $nbElem < K$ **faire**
14 $\quad iter \mathrel{+}= 1$;
15 $\quad \Delta = min(K - nbElem, cards[iter])$
16 $\quad somme \mathrel{+}= 2^{\Delta - |c_1|}$
17 $\quad nbElem \mathrel{+}= \Delta$
18 **fin**
19 **return** $c_1 + log_2(somme)$

de toutes les cliques restantes, en enlevant tous les sommets déjà sélectionnés dans le sous-graphe de la solution courante. Après avoir trié les cardinaux dans l'ordre décroissant à la ligne 12, l'algorithme ajoute successivement les cliques sans vérifier la contrainte de disjonctions, jusqu'à avoir atteint la limite du nombre de sommets K. Avec l'algorithme de calcul du logarithme de l'objectif et le calcul de borne supérieure d'une solution partielle, nous pouvons définir un algorithme *Branch and Bound* pour résoudre notre modèle M_2.

4.5.3 Algorithme *Branch & Bound*

En utilisant l'Algorithme 2 ($objective$) et l'Algorithme 3 ($upperBoundM_2$), nous définissions l'algorithme *Branch & Bound* du modèle M_2 (BBM_2) : l'Algorithme 4.

Algorithme 4 : $BBM_2(L, K, current, index, lowerBound)$

Entrées : L : Liste des cliques maximales
K : Taille limite du sous-graphe
current : Liste de cliques disjointes, formant la solution courante
index : Entier indiquant la position courante dans la liste L
lowerBound : Meilleure valeur objective trouvée
Résultat : Liste de cliques disjointes couvrant le sous-graphe et la sécurité associée

```
1  si upperBoundM2(L, current, K, index) ≤ lowerBound alors
   |  /* Bounded solution                                          */
2  fin
3  newIndex ← findNextClique(L, current, index)
4  security ← objective(solution)
5  solution ← current
6  vertices ← ⋃_{c∈current} c
7  pour tous les c ∈ P( L[newIndex] \ vertices ) faire
8  |  si |current ∪ c| ≤ K alors
9  |  |  lsol, lsec ← BBM2(L, K, append(current, c), newIndex, lowerBound)
10 |  |  si lsec > security alors
11 |  |  |  solution ← lsol
12 |  |  |  security ← lsec
13 |  |  |  si security > lowerBound alors
14 |  |  |  |  lowerBound ← security
15 |  |  |  fin
16 |  |  fin
17 |  fin
18 fin
19 return solution, security
```

Le premier appel de la fonction est $BBM_2(L, K, emptyList(), 0, 0)$ avec L la liste des cliques maximales, et K la taille du sous-graphe (i.e. le nombre de portes clés à ajouter). La ligne 1 teste si la borne supérieure de la partition courrante $current$ est plus petite que la meilleur solution trouvée $lowerBound$. Si c'est le cas, alors aucune solution ayant $current$ en sous-partie n'est solution optimale. Explorer cette branche de l'arbre de recherche est inutil. La ligne 3 cherche la prochaine clique dans la liste qui possède un sommet différent de ceux déjà présents dans $current$. La fonction $findNextClique$ retourne l'indice de cette clique. La gestion des indices de cliques avec $index$ et $newIndex$ permet de ne jamais calculer deux fois la même solution. La boucle à la ligne 7 permet de brancher sur tous les sous-ensembles de la clique dans $L[newIndex]$, assurant ainsi une exploration totale de l'espace de recherche. La condition de la ligne 8 assure que la contrainte de sac à dos (contrainte 4.21)

soit respectée. L'appel récursif à la ligne 9 permet d'explorer récursivement l'arbre de recherche. Enfin les lignes 10 à 16 permettent de mémoriser la meilleur solution trouvée.

L'algorithme BBM_2 permet d'explorer toutes les solutions possibles, s'assurant ainsi que l'algorithme trouvera toujours la solution optimale en un temps fini. Néanmoins ce temps peut être long, nous proposons alors une optimisation du temps de calcul en appliquant une stratégie de résolution itérative ci-dessous.

4.5.4 Stratégie de résolution itérative

La liste L contient un nombre exponentiel de cliques, soit $\mathcal{O}(3^{\frac{n}{3}})$. Nous proposons de calculer la solution optimale avec une sous-liste de L. Cette solution n'a aucune garantie d'être la solution optimale pour la liste complète L, nous la dénommons solution sous-optimale. Avec un bon choix de sous-liste, la solution sous-optimale est probablement une solution qui a une valeur objective proche de l'optimale, permettant ainsi de couper l'arbre de recherche grâce aux calculs de bornes de l'Algorithme 3. Pour trouver une solution optimale du problème, il faut calculer des solutions sous-optimales en complétant progressivement la liste réduite à chaque itération, jusqu'à obtenir la liste complète. Notons que l'on peut également arrêter les itérations lorsque la borne supérieure globale est atteinte ou lorsque les cliques supplémentaires sont plus petites que la plus petite clique de la partition trouvée. Dans l'analyse des résultats qui suit, cette méthode de résolution est appelée "loop".

4.5.5 Résultats numériques

Nous voulons comparer les résultats du solveur $Gurobi$ [Gur23] sur le modèle M_2 avec l'algorithme *Branch and Bound*. Les circuits proviennent du *benchmark* $ISCAS-85$ [BF85], très utilisé dans la communauté électronique. L'implémentation des algorithmes est en Julia 1.8 [Bez+17], et les calculs ont été effectué sur un Windows 10 64 bits, Intel $i7-8665U$ avec 16 Go de mémoire.

La Tableau 4.4 compare une résolution par $gurobi$ du problème M_2, avec l'Algorithme 4. Le tableau contient les informations des circuits, et les temps de résolution pour $Gurobi$ et pour l'Algorithme 4. Pour chaque résolution, nous proposons une version itérative (*loop*) présentée dans la Section 4.5.4. La résolution prend en compte le temps de création du modèle pour la résolution par solveur. Comme le temps de calcul de la liste de cliques maximales est identique quelque soit la résolution, il n'est

Information du circuit				Modèle M_2 (Gurobi)		Algorithme BBM_2	
circuit	#sommets	#portes clés	sécurité optimale	Temps optimisation (s)	Temps optimisation loop(s)	Temps optimisation (s)	Temps optimisation loop(s)
c499	207	10 (5%)	$2^{10.0}$	94.33	0.45	0.25	0.06
		21 (10%)	$2^{21.0}$	133.66	0.53	0.22	0.12
		31 (15%)	$2^{31.0}$	112.33	0.58	0.22	0.07
c880	228	11 (5%)	$2^{7.1699}$	0.15	0.09	0.03	0.02
		23 (10%)	$2^{8.5999}$	0.07	0.04	0.09	0.16
		34 (15%)	$2^{9.1699}$	0.12	0.04	0.06	0.11
c1355	207	10 (5%)	$2^{10.0}$	43.26	0.15	0.07	0.03
		21 (10%)	$2^{21.0}$	203.75	0.2	0.07	0.06
		31 (15%)	$2^{31.0}$	27.57	0.27	0.07	0.03
c3540	491	25 (5%)	$2^{25.0}$	0.15	0.05	0.001	0.001
		49 (10%)	$2^{49.0}$	0.18	0.06	0.001	0.01
		74 (15%)	$2^{74.0}$	-	-	0.001	0.001
c5315	758	38 (5%)	$2^{38.0}$	0.15	0.07	0.001	0.001
		76 (10%)	2^{71}	-	-	0.001	0.01
		114 (15%)	2^{71}	-	-	0.001	3.74
c6288	1480	74 (5%)	$2^{27.0000}$	0.41	0.4	600*	1.89
		148 (10%)	$2^{27.0000}$	0.31	0.65	600*	1.28
		222 (15%)	$2^{27.0000}$	0.32	0.84	600*	0.67
c7552	949	47 (5%)	$2^{47.0}$	0.23	0.09	0.001	0.001
		95 (10%)	$2^{95.0}$	-	-	0.001	0.001
		142 (15%)	$2^{142.0}$	-	-	0.01	0.001

Tableau 4.4. : Comparaison du temps de résolution du modèle M_2 avec $Gurobi$ et l'Algorithme 4 sur les circuit $ISCAS-85$

pas pris en compte dans le temps de résolution. Les cases vides $(-)$ dans le tableau correspondent à une résolution incohérente décrite précédemment. Nous constatons que les versions itératives sont toujours au moins aussi rapides à résoudre que les versions complètes. Dans le cas du circuit $c6288$, la version itérative du *Branch & Bound* trouve la solution optimale rapidement, contrairement à la version complète. Cette version itérative permet de réduire considérablement l'espace de recherche. Ce résultat était donc attendu.

Nous constatons que notre algorithme *Branch & Bound* permet une résolution exacte sur ces instances de référence de la communauté. Nous souhaitons connaitre la possibilité de l'existence d'un algorithme d'approximation en temps polynomial pour notre problème. La section suivante discute de l'existence d'un tel algorithme.

4.5.6 Complexité d'approximation du problème WCC

Dans la Section 4.3.3, nous avons prouvé que le problème est $NP\text{-}DUR$. Nous voulons alors savoir s'il existe un algorithme d'approximation pour le problème WCC. En reprenant le raisonnement de la preuve de complexité de la preuve de $NP\text{-}DUR$, nous allons démontrer que le problème WCC est $NON\text{-}APX$.

Pour cette preuve, nous utilisons les notations suivantes :

— $G = (V, E)$: un graphe avec $n = |V|$: la taille du graphe
— $z(P) = \sum_{c \in P} 2^{|c|}$: la valeur objective de la partition en cliques P du problème WCC.
— $\omega(G)$: la taille de la plus grande clique dans le graphe G
— $\omega(P)$: la taille de la plus grande clique dans la partition en cliques P
— $A^*(G)$: un algorithme de résolution exacte du problème WCC
— P^* : une partition en cliques optimale pour le problème WCC, retournée par $A^*(G)$.
— $\tilde{A}(G)$: un algorithme d'approximation avec un ratio constant r du problème WCC
— $\tilde{P}$: une partition en cliques retournée par $\tilde{A}(G)$

Afin de démontrer que le problème WCC est $NON\text{-}APX$, nous allons démontrer que :

1. La plus grande clique d'une partition approchée $\tilde{P}$, résultat d'un algorithme d'approximation $\tilde{A}(G)$ avec un ratio r constant du problème WCC, permet d'encadrer la taille de la plus grande clique du graphe.
2. Cet encadrement ne peut pas être obtenu avec un algorithme polynomial, donc l'algorithmes $\tilde{A}(G)$ n'est pas polynomial.

Encadrement

En utilisant les lemmes de la section Section 4.3.3, nous allons prouver le résultat d'encadrement.

Théorème 4.5.1. *Soit $G = (V, E)$ un graphe, et soit $n = |V|$. Soit P^* la partition en cliques optimale résultat de $A^*(G)$. Soit $\omega(G)$ la taille de la plus grande clique dans G. Soit $\tilde{A}(G)$ un algorithme d'approximation pour WCC avec un ratio constant $r \geq 1$ tel que l'algorithme renvoie une partition $\tilde{P}$ satisfaisant $z(\tilde{P}) \leq z(P^*) \leq r \times z(\tilde{P})$. Soit $\omega(\tilde{P})$ la taille de la plus grande clique dans $\tilde{P}$. L'inégalité suivante est vérifiée :* $\omega(\tilde{P}) \leq \omega(G) \leq \omega(\tilde{P}) + \log_2(n)$.

Démonstration. Commençons par reformuler les définitions de la preuve précédente. Soit un algorithme d'approximation $\tilde{A}(G)$ avec un ratio constant r tel que l'algorithme renvoie une partition $\tilde{P}$ satisfaisant $z(\tilde{P}) \leq z(P^*) \leq r \times z(\tilde{P})$. Soit $\overline{P}$ la meilleure solution contenant une clique de taille $\omega(G)$. Par définition de l'optimalité, nous avons $z(\overline{P}) \leq z(P^*) \leq r \times z(\tilde{P})$. Considérons deux cas :

1. $\omega(\tilde{P}) \geq \omega(P^*)$. Selon le Théorème 4.3.5, $\omega(G) - \log_2(n) \leq \omega(P^*) \leq \omega(G)$ implique $\omega(G) - \log_2(n) \leq \omega(P^*) \leq \omega(\tilde{P}) \leq \omega(G)$.
2. $\omega(\tilde{P}) \leq \omega(P^*) \leq \omega(G)$. Cette condition peut s'exprimer comme $\exists \tilde{\Delta} \in \mathbb{N}, \omega(\tilde{P}) + \tilde{\Delta} = \omega(P^*)$. De plus, $\exists \Delta \in \mathbb{N}, \omega(\tilde{P}) + \tilde{\Delta} + \Delta = \omega(P^*) + \Delta = \omega(G)$. De plus, $r \times z(\tilde{P}) \geq z(P^*) \geq z(\overline{P})$ implique $r \times z(\tilde{P}) \geq z(\overline{P})$. Pour compenser la différence d'objectif entre $2^{\omega(G)}$ et $\frac{1}{r}2^{\omega(\tilde{P})}$, nous avons besoin d'une valeur d'objectif d'au moins $r \times 2^{\tilde{\Delta}+\Delta} \times 2^{\omega(\tilde{P})}$, ce qui nécessite au minimum $r \times 2^{\tilde{\Delta}+\Delta} \times \omega(\tilde{P})$ sommets, selon le Lemme 4.3.4. Cependant, le nombre de sommets est limité à n, donc nous avons :

$$
\begin{aligned}
& r \times 2^{\tilde{\Delta}+\Delta} \times \omega(\tilde{P}) \leq n \\
\iff & 2^{\tilde{\Delta}+\Delta} \leq \frac{n}{r \times \omega(\tilde{P})} \\
\iff & \tilde{\Delta} + \Delta \leq \log_2\left(\frac{n}{r \times \omega(\tilde{P})}\right) \\
\iff & \omega(G) - \omega(\tilde{P}) \leq \log_2(n) - \log_2(r) - \log_2(\omega(\tilde{P})) \\
\iff & \omega(G) - \log_2(n) \leq \omega(\tilde{P}) - \log_2(r) - \log_2(\omega(\tilde{P})) \\
\iff & \omega(G) - \log_2(n) \leq \omega(\tilde{P}) \qquad \text{puisque } r \geq 1 \implies \log_2(r) \geq 0
\end{aligned}
$$

Puisque $\omega(\tilde{P}) \leq \omega(G)$ par définition, nous concluons que :

$$
\omega(G) - \log_2(n) \leq \omega(\tilde{P}) \leq \omega(G) \iff \omega(\tilde{P}) \leq \omega(G) \leq \omega(\tilde{P}) + \log_2(n)
$$

Par conséquent, la solution d'approximation utilisant un ratio constant r pour WCC fournit une approximation de $\omega(G)$ à un écart de $\log_2(n)$ près. □

NON-APX

Enfin prouvons que le problème WCC est NON-APX.

Théorème 4.5.2. *Soit $\tilde{A}(G)$ un algorithme d'approximation de WCC avec un ratio constant $r \geq 1$. Si $P \neq NP$, pour tous les graphes $G = (V, E)$ tels que $|V| \geq 16$, $\tilde{A}(G)$ n'est pas polynomial. Le problème WCC est donc NON-APX.*

Démonstration. Soit $G = (V, E)$ un graphe, et soit $n = |V|$. Soit $\tilde{A}(G)$ un algorithme d'approximation de WCC avec un ratio constant $r \geq 1$. Soit $\tilde{P}$ la partition en cliques approché résultat de $\tilde{A}(G)$ telle que $z(\tilde{P}) \leq z(P^*) \leq r \times z(\tilde{P})$. Soit $\omega(G)$ la taille de la plus grande clique dans G, et $\omega(\tilde{P})$ la taille de la plus grande clique dans $\tilde{P}$. Selon le Théorème 4.5.1, $\tilde{A}(G)$ permet de trouver une borne de $\omega(G)$:

$$
\begin{aligned}
& \omega(\tilde{P}) \leq \omega(G) \leq \omega(\tilde{P}) + \log_2(n) \\
\iff & \omega(\tilde{P}) \leq \omega(G) \leq \omega(\tilde{P}) \times \log_2(n) && \text{(Pour } n \geq 4\text{, voir le Lemme 4.3.2)} \\
\iff & \omega(\tilde{P}) \leq \omega(G) \leq \omega(\tilde{P}) \times \sqrt{n} = \omega(\tilde{P}) \times n^{\frac{1}{2}} && \text{(Pour } n \geq 16\text{, voir le Lemme 4.3.3)}
\end{aligned}
$$

Cependant, selon le Théorème 4.3.1, il n'existe pas d'algorithme polynomial pour approximer $\omega(G)$ avec un ratio inférieur à $n^{1-\epsilon}$ pour tout $\epsilon > 0$, en particulier pour $\epsilon = \frac{1}{2}$. Par conséquent, $\tilde{A}(G)$ n'est pas polynomial, il n'existe donc pas d'algorithme polynomial pour trouver une solution approchée de WCC avec un ratio constant $r \geq 1$. Ainsi, le problème WCC est NP-DUR. □

Grâce au Théorème 4.5.2, nous avons prouvé que le problème WCC est $NON\text{-}APX$. Le problème WCC étant un cas particulier de notre problème de sécurité avec $K = n$, si nous voulons par la suite utiliser des algorithmes de résolution non optimale, nous devrons utiliser des heuristiques ou des méta-heuristiques sans garantie de ratio d'approximation.

4.6 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle approche du problème SLL. Cette nouvelle approche nous a permis de modéliser le problème de sécurité avec deux modèles M_1 et M_2. Le modèle M_2 nécessite un pré-traitement exponentiel, mais comme le problème est $NP\text{-}DUR$, la complexité temporelle n'est pas modifiée. En pratique, en comptant le prétraitement exponentiel, le modèle M_2 est plus rapide et résoud plus d'instances. Nous avons ensuite proposé un algorithme *Branch & Bound* pour résoudre le modèle M_2. Cet algorithme s'avère efficace grâce à un bon calcul de bornes et à notre stratégie de résolution itérative. Nous avons enfin prouvé l'inapproximabilité du problème WCC, et par conséquent, celle du problème de sécurité. Les résultats de l'algorithme *Branch & Bound* ont été publiés dans la 9^{th} International Conference on Control, Decision and Information Technologies (CoDIT) [Fon+23].

5 Intégration du Logic Locking dans le flot de conception

Sommaire

5.1 Introduction

Dans ce chapitre, nous souhaitons intégrer le *logic locking* dans le flot de conception. Nous souhaitons concevoir une extension de sécurité basée sur nos modèles développés dans le chapitre précédent. Nous visons également à limiter l'impact sur le délai, déterminant les performances du circuit. Nous étudions alors l'optimisation du délai pour l'insertion de portes clés dans le circuit. Nous regroupons ensuite les travaux sur l'optimisation de la sécurité et du délai, obtenant ainsi un modèle de sécurité avec une contrainte temporelle et un modèle multi-objectifs. Nous proposons finalement une extension de sécurité pour le flot de conception, exploitant les travaux de cette thèse et les meilleurs algorithmes proposés. Nous examinons les résultats sur deux groupes de *benchmarks*, puis nous nous comparons à l'article initial sur le *Strong Logic Locking (SLL)* [Yas+16b].

5.2 Modélisation temporelle

Les délais dans un circuit sont dus aux délais de propagation des portes logiques et aux délais de propagation des fils. Ajouter des portes dans le circuit va alors nécessairement augmenter les délais. Cependant, seul le plus long chemin (en termes de délai) compte. Il est appelé **chemin critique**. Les performances du circuit sont inversement proportionnelles à la taille du chemin critique. Nous cherchons donc à minimiser le plus long chemin en insérant des portes logiques.

Nous souhaitons créer un modèle mathématique qui représente l'ajout de portes logiques dans le circuit. Pour modéliser le problème, nous utilisons les notations suivantes :

— $G = (V, E)$: la représentation en graphe de la *netlist* du circuit avec V les portes logiques et E les fils

— $inputs(G)$: les entrées du circuit G

— $outputs(G)$: les sorties du circuit G

— $\delta^{-}(g)$: les sommets précédant la porte g

On suppose que chaque porte clé a le même temps de propagation.

Supposons la position d'insertion représentée par la Figure 5.1. En ajoutant une porte clé à la suite de $G1$, on obtient la Figure 5.2. Constatons que, pour ajouter une porte, il faut conserver l'ancien fil en bleu, ajouter un fil rouge entre la porte logique et la porte clé, un autre fil rouge entre la porte clé et la clé numérique, et enfin, ajouter une porte clé, dans notre cas un XOR ou un $XNOR$.

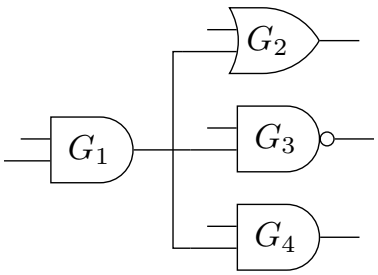


Figure 5.1. : Position d'insertion dans un circuit

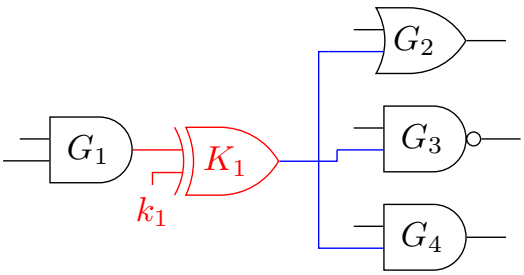


Figure 5.2. : Insertion d'une porte clé dans un circuit

En supposant qu'un fil d'insertion aura la même longueur, quelle que soit la position, ainsi que la même longueur entre une porte clé et la clé numérique, on peut estimer cet ajout comme constant et l'ajouter au délai d'une porte clé.

5.2.1 Modélisation avec portes et fils

Pour modéliser le problème, notons les données suivantes :

— $G = (V, E)$: la représentation *netlist* du circuit avec V les portes logiques et E les fils

— T_g : le temps de propagation de la porte g

— T_{key} : le temps de propagation d'une porte clé et des fils d'ajout, supposé constant

— $T_{net(g,g')}$: le délai du net entre la porte g et g' Cette information peut être obtenue suite à un premier placement-routage

Notons la variable de décision : $x_g = \begin{cases} 1 & \text{si la porte } g \text{ est suivie d'une porte clé} \\ 0 & \text{sinon} \end{cases}$

Notons enfin les variables de modélisation :

— d_g : le délai du cône logique (vu dans la Section 2.2) de g

— d_{max} : le délai maximum du circuit (la valeur du chemin critique)

L'objectif (5.1) est de minimiser le délai maximum du circuit, donc de minimiser la valeur du chemin critique.

$$\min \quad D_{max} \tag{5.1}$$

La contrainte d'insertion (5.2) impose un nombre K de portes clés à ajouter dans le circuit.

$$\sum_{g \in V} x_g = K \tag{5.2}$$

La contrainte d'initialisation (5.3) impose que les entrées soient à 0, sauf si une porte clé y est insérée.

$$d_i = 0 + x_i \times T_{key} \qquad \forall i \in inputs(G) \tag{5.3}$$

La contrainte de propagation (5.4a) impose que le délai à la porte g soit le temps de propagation de la porte g avec le temps de la porte clé si une insertion a lieu, ajoutée au plus long chemin entre les entrées et cette porte. La contrainte (5.4b) est

une forme linéaire de cette contrainte. Comme l'objectif est de minimiser le plus long chemin, la linéarisation est valide.

$$d_g = T_g + x_g \times T_{key} + \max_{g' \in \delta^-(g)} \left\{ d_g + T_{net(g',g)} \right\} \qquad \forall g \in V \backslash inputs(G) \qquad (5.4a)$$

$$d_g \geq T_g + x_g * T_{key} + d_g + T_{net(g',g)} \qquad \forall g \in V,\ \forall g' \in \delta^-(g) \qquad (5.4b)$$

La contrainte du plus grand chemin (5.5a) impose que le plus long chemin soit le max des plus longs chemins pour aller d'une entrée à la sortie, ce qui est la définition du chemin critique. La contrainte (5.5b) est une forme linéaire de cette contrainte. Cette linéarisation fonctionne car l'objectif minimise le plus long chemin.

$$d_{max} = \max_{o \in outputs(G)} d_o \qquad (5.5a)$$

$$d_{max} \geq \quad d_o \qquad \forall o \in outputs(G) \qquad (5.5b)$$

Ces contraintes permettent de construire le modèle M_3 qui possède $\mathcal{O}\left(2 * |V| + 1\right)$ variables et $\mathcal{O}\left(|V| + |E|\right)$ contraintes.

$$M_3 = \begin{cases} \min & d_{max} & & (5.1) \\ \underline{s.t.} & \sum_{g \in V} x_g = K & & (5.2) \\ & d_i = x_i * T_{key} & \forall i \in inputs(G) & (5.3) \\ & d_g \geq T_g + x_g * T_{key} + d_g + T_{net(g',g)} & \forall g \in V \backslash inputs(G),\ \forall g' \in \delta^-(g) & (5.4) \\ & d_{max} \geq d_o & \forall o \in outputs(G) & (5.5) \\ & d_{max} \geq 0,\ d_g \geq 0 & \forall g \in V & (5.6) \\ & x_g \in \{0,1\} \quad \forall g \in V & & (5.7) \end{cases}$$

Remarque

Pour les technologies de gravure de plus de 100nm, l'importance des délais réside majoritairement dans les portes logiques, et le délai des fils est considéré comme négligeable. Ceci est le cas des technologies de gravure ouvertes et matures. Nous proposons donc une simplification de notre modèle en ignorant le délai des fils, $T_{net(g',g)} = 0$, que nous présentons dans la section suivante.

5.2.2 Modélisation sans fil

Pour modéliser notre problème sans fil, nous allons réutiliser les données et variables précédentes : Pour modéliser le problème, notons la variable de décision :

— $x_g = \begin{cases} 1 & \text{la porte } g \text{ est suivie d'une porte clé} \\ 0 & \text{sinon} \end{cases}$

Notons les variables de modélisation :

— d_g : le délai du cone logique (vu dans la Section 2.2) de g

— d_{max} : le délai maximum du circuit (la valeur du chemin critique).

Notons enfin les données du problèmes :

— T_g : le temps de propagation de la porte g

— T_{key} le temps de propagation d'une porte clé et sans les fils d'ajout

Notons que le T_{key} ne prend plus en compte les fils d'ajout supposé dans la Section 5.2. L'objectif (5.1) et les contraintes (5.2), (5.3) et (5.5) sont identiques et sont les équations (5.8) - (5.10) et (5.12). L'équation (5.4b) devient l'équation (5.11) ci-dessous :

$$d_g \geq T_g + x_g * T_{key} + d_g \qquad \forall g \in V,\ \forall g' \in \delta^-(g) \qquad (5.11)$$

Ces contraintes permettent de construire le modèle M_4 qui possèdent $\mathcal{O}(2 * |V| + 1)$ variables et $\mathcal{O}(|V| + |E|)$ contraintes.

$$M_4 = \begin{cases} \min & d_{max} & & (5.8) \\ \underline{s.t.} & \sum_{g \in V} x_g = K & & (5.9) \\ & d_i = x_i * T_{key} & \forall i \in inputs(G) & (5.10) \\ & d_g \geq T_g + x_g * T_{key} + d_{g'} & \forall g \in V \backslash inputs(G),\ \forall g' \in \delta^-(g) & (5.11) \\ & d_{max} \geq d_o & \forall o \in outputs(G) & (5.12) \\ & d_{max} \geq 0,\ d_g \geq 0 & \forall g \in V & (5.13) \\ & x_g \in \{0,1\} \quad \forall g \in V & & (5.14) \end{cases}$$

5.2.3 Résultats numériques

Observons les résultats que l'on obtient en cherchant à minimiser le chemin critique. Nous utilisons la *sxlib* qui est une librairie de cellule symbolique. Nous allons mesurer le résultat du modèle M_4. Les circuits utilisés sont obtenus avec les benchs $ISCAS-85$ [BF85], le code est implémenté en $Julia 1.8$ [Bez+17], et le modèle M_4 est résolu avec $Gurobi$ 9.11 [Gur23]. Les calculs ont été effectués sur un Windows 10 64 bits, Intel $i7-8665U$ avec 16 Go de mémoire.

Information du circuit			Optimisation de M_4 avec Gurobi			*B&B*	
circuit	#sommets	#portes clés	Temps optimisation (s)	Sécurité	Délai (%)	Sécurité	Délai (%)
c499	207	10 (5%)	0.69	$2^{4.8074}$	0.0	$2^{10.0}$	9.76
		20 (10%)	0.63	$2^{5.4594}$	0.0	$2^{21.0}$	19.52
		31 (15%)	0.65	$2^{6.4263}$	0.0	$2^{31.0}$	29.28
c880	228	11 (5%)	0.75	$2^{4.4594}$	0.0	$2^{7.1699}$	0.0
		23 (10%)	0.73	$2^{5.5236}$	0.0	$2^{8.5999}$	0.0
		34 (15%)	0.66	$2^{6.0875}$	0.0	$2^{9.1699}$	4.33
c1355	207	10 (5%)	0.51	$2^{4.8074}$	0.0	$2^{10.0}$	9.76
		21 (10%)	0.47	$2^{5.5236}$	0.0	$2^{21.0}$	19.52
		31 (15%)	0.44	$2^{6.4594}$	0.0	$2^{31.0}$	29.29
c3540	491	25 (5%)	1.83	$2^{6.7814}$	0.0	$2^{25.0}$	3.51
		49 (10%)	2.17	$2^{11.1215}$	0.0	$2^{49.0}$	3.51
		74 (15%)	1.58	$2^{18.0013}$	0.0	$2^{74.0}$	2.56
c5315	758	38 (5%)	2.29	$2^{6.2479}$	0.0	$2^{38.0}$	5.14
		76 (10%)	2.43	$2^{10.1774}$	0.0	$2^{71.0}$	5.14
		114 (15%)	2.08	$2^{8.8948}$	0.0	$2^{71.0}$	10.48
c6288	1480	74 (5%)	4.29	$2^{7.2288}$	0.0	$2^{27.0000}$	12.34
		148 (10%)	4.13	$2^{8.2668}$	0.0	$2^{27.0000}$	22.55
		222 (15%)	4.12	$2^{8.8517}$	0.0	$2^{27.0000}$	22.55
c7552	949	47 (5%)	2.86	$2^{14.0060}$	0.0	$2^{47.0}$	7.09
		95 (10%)	2.66	$2^{34.0000}$	0.0	$2^{95.0}$	9.29
		142 (15%)	2.63	$2^{59.0}$	0.0	$2^{142.0}$	17.88

Tableau 5.1. : Résultats du modèle M_4 sur les circuit $ISCAS-85$

La Tableau 5.1 agrège les résultats du modèle M_4 obtenus sur les benchs $ISCAS-85$, en comparant la sécurité et le délai obtenus. Le tableau est composé de trois blocs de colonnes, le premier bloc comporte les informations sur les circuits, le deuxième le temps et les résultats du modèle M_4 résolu avec le solveur $Gurobi$ et le troisième agrège les résultats de temps et sécurité obtenus par l'Algorithme 4 de la Section 4.5.3. Nous constatons que ce modèle est résolu par le solveur en moins de 5 secondes et que l'on obtient une augmentation du délai de 0% dans tous les cas. Néanmoins, comme attendu, la sécurité obtenue est très faible comparée aux solutions obtenues par la résolution du modèle M_2. La grande disparité entre les solutions obtenues en

minimisant le délai et en maximisant la sécurité nous laisse supposer qu'il existe une grande variété de solutions avec des compromis entre les deux objectifs.

5.3 Fusion des modèles temporels et de sécurité

Nous avons exploré deux approches pour fusionner les travaux sur la sécurité et les délais. La première consiste à ajouter une contrainte temporelle dans un modèle de sécurité. La deuxième consiste à proposer une modélisation multi-objectifs.

5.3.1 Intégration d'une contrainte temporelle dans le modèle de sécurité

Il est possible de contraindre le modèle de sécurité pour prendre en compte le délai. Comme nous voulons réduire le délai du chemin critique, nous proposons d'interdire une insertion sur le chemin critique. Pour cela, nous allons en premier lieu calculer le chemin critique en ne considérant que le délai des portes.

Notons Π l'ensemble des sommets composant le chemin critique. Nous pourrions imposer une contrainte dans le modèle M_2 :

$$x_v = 0 \quad \forall v \in \Pi$$

Cependant, nous pouvons simplement supprimer ces sommets du GIC (Section 4.2) en pré-traitement, sans rajouter de contrainte au modèle. Cela permet également d'utiliser les calculs de bornes de sécurité (Algorithme 3).

Le Tableau 5.2 contient les informations des circuits, les résultats en sécurité et en surcoût de délai pour le modèle M_4, l'Algorithme Branch & Bound M_2 sans (BB) et avec la contrainte de chemin critique ($BB + S$). Les temps de calcul entre BB et $BB + S$ sont équivalents, cette donnée n'est pas affichée dans ce tableau. Les Figures (5.3) et (5.4) représentent ces données sous forme de diagrammes. En absysse, nous avons les circuit avec les variations du nombre de portes clés : $5\%, 10\%, 15\%$ de la taille totale du circuit. En ordonné, nous avons le surcoût de délai (en %), plus ce surcoût approche 0, plus la solution est souhaitable. La barre verte correspond aux données du modèle M_4, la barre bleu correspond aux données du BB et la barre orange aux données de $BB + S$.

Dans nos résultats, on constate que dans tous les cas, la sécurité du modèle M_4 est inférieure à celle de $BB + S$, qui est lui-même inférieur ou égal à BB. Ce résultat est attendu, puisque le modèle BB cherche la meilleure sécurité, le modèle $BB + S$ cherche la meilleure sécurité sans s'insérer dans le chemin critique, et que le modèle M_4 minimise le surcoût de délai, indépendamment de la sécurité. Pour tous les circuits, la sécurité de BB est équivalente à celle de $BB + S$. Le surcoût de délai est diminué dans 52.38% des cas, et inchangé pour 33.33% des cas. La stratégie permet de réduire le délai dans 52.38% des cas. Elle s'aggrave dans 14.28% des cas. En supprimant les chemins critiques, l'algorithme cherche à trouver des positions d'insertion qui maximisent la sécurité, et ces positions peuvent être sur des chemins presque critiques.

Information du circuit			M_4 (Gurobi)		*BB*		*BB+S*	
circuit	#sommets	#portes clés	Sécurité	Délai (%)	Sécurité	Délai (%)	Sécurité	Délai (%)
c499	207	10 (5%)	$2^{4.8074}$	0.0	$2^{10.0}$	9.76	$2^{10.0}$	4.85
		20 (10%)	$2^{5.4594}$	0.0	$2^{21.0}$	19.52	$2^{21.0}$	19.52
		31 (15%)	$2^{6.4263}$	0.0	$2^{31.0}$	29.28	$2^{31.0}$	19.52
c880	228	11 (5%)	$2^{4.4594}$	0.0	$2^{7.1699}$	0.0	$2^{7.1699}$	0.0
		23 (10%)	$2^{5.5236}$	0.0	$2^{8.5999}$	0.0	$2^{8.5999}$	4.33
		34 (15%)	$2^{6.0875}$	0.0	$2^{9.1699}$	4.33	$2^{9.1699}$	4.33
c1355	207	10 (5%)	$2^{4.8074}$	0.0	$2^{10.0}$	9.76	$2^{10.0}$	9.76
		21 (10%)	$2^{5.5236}$	0.0	$2^{21.0}$	19.52	$2^{21.0}$	19.52
		31 (15%)	$2^{6.4594}$	0.0	$2^{31.0}$	29.29	$2^{31.0}$	19.52
c3540	491	25 (5%)	$2^{6.7814}$	0.0	$2^{25.0}$	3.51	$2^{25.0}$	2.01
		49 (10%)	$2^{11.1215}$	0.0	$2^{49.0}$	3.51	$2^{49.0}$	4.34
		74 (15%)	$2^{18.0013}$	0.0	$2^{74.0}$	2.56	$2^{74.0}$	2.56
c5315	758	38 (5%)	$2^{6.2479}$	0.0	$2^{38.0}$	5.14	$2^{38.0}$	0.0
		76 (10%)	$2^{10.1774}$	0.0	$2^{71.0}$	5.14	$2^{71.0}$	5.14
		113 (15%)	$2^{8.8948}$	0.0	$2^{71.0}$	0.0	$2^{71.0}$	5.14
c6288	1480	74 (5%)	$2^{7.2288}$	0.0	$2^{27.0000}$	10.0	$2^{27.0000}$	5.63
		148 (10%)	$2^{8.2668}$	0.0	$2^{27.0000}$	15.0	$2^{27.0000}$	10.1
		222 (15%)	$2^{8.8517}$	0.0	$2^{27.0000}$	20.0	$2^{27.0000}$	13.78
c7552	759	47 (5%)	$2^{14.0060}$	0.0	$2^{47.0}$	12.5	$2^{47.0}$	7.09
		95 (10%)	$2^{34.0000}$	0.0	$2^{95.0}$	17.5	$2^{95.0}$	0.71
		142 (15%)	$2^{59.0000}$	0.0	$2^{142.0}$	22.5	$2^{142.0}$	9.29

Tableau 5.2. : Sécurité et délai obtenus après une résolution optimale du modèle M_4, du Branch & Bound sans et avec la stratégie d'optimisation du délai

Cette diversité de solutions nous laisse penser qu'il existe de nombreux points dans le front de Pareto du problème. Des algorithmes d'optimisation multi-objectifs nous permettraient alors de prendre en compte les deux objectifs et de générer plusieurs solutions avec des compromis divers.

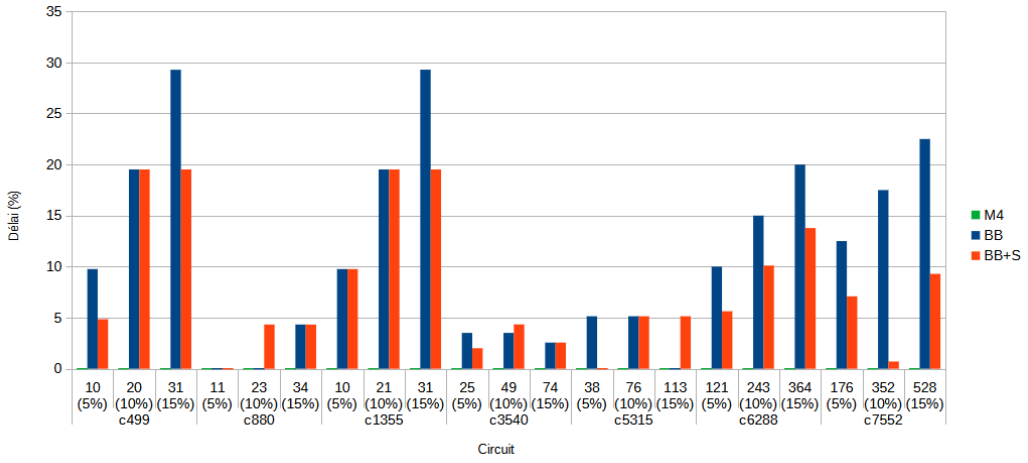


Figure 5.3. : Comparaison des délais du modèle M_4, du Branch & Bound sans et avec la stratégie d'optimisation du délai

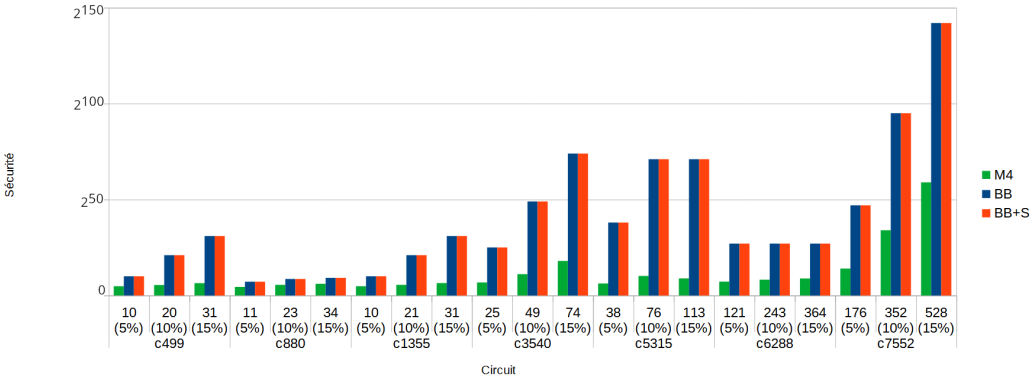


Figure 5.4. : Comparaison de la sécurité du modèle M_4, du Branch & Bound sans et avec la stratégie d'optimisation du délai

5.3.2 Modélisation multi-objectives

Nous proposons une modélisation du problème multi-objectifs. Ce modèle est une fusion du modèle M_2 et du modèle M_4. Les données sont :

- $G = (V, E)$: un graphe représentant la *netlist* avec V les portes logiques et E les fils
- $G' = (V, E')$: le graphe d'interférence complet (Section 4.2) de G
- K : le nombre de portes logiques à ajouter dans le circuit
- T_g : le temps de propagation de la porte g
- T_{key} : le temps de propagation d'une porte clé et des fils d'ajout, supposé constant
- $T_{net(g,g')}$: le délai du net entre la porte g et g'

Nous gardons la même variable de décisions : $x_g = \begin{cases} 1 & \text{si la porte } g \text{ est suivie d'une porte clé} \\ 0 & \text{sinon} \end{cases}$

Notons les variables de modélisation :

- w_c : la taille de la sous-clique c sélectionnée
- $\lambda_c = \begin{cases} 1 & \text{si la clique } c \text{ est sélectionnée} \\ 0 & \text{sinon} \end{cases}$
- $a_c^g = \begin{cases} 1 & \text{si la porte } g \text{ est sélectionnée dans la clique } c \\ 0 & \text{sinon} \end{cases}$
- d_g : le délai du cône logique (vu dans la Section 2.2) de la porte g
- d_{max} : le délai maximum du circuit (la valeur du chemin critique)

Nous obtenons alors le modèle M_5 qui possède $\mathcal{O}(|V| * |L|)$ variables et $\mathcal{O}(|V| * |L|)$ contraintes.

$$
M_5 = \begin{cases}
\max & \displaystyle\sum_{c \in L} 2^{w_c} * \lambda_c & & (5.15) \\
\min & d_{max} & & (5.16) \\
\underline{s.t.} & \displaystyle\sum_{g \in V} x_g \leq K & & (5.17) \\
& w_c = \displaystyle\sum_{g \in C} a_c^g & \forall c \in L & (5.18) \\
& \lambda_c \leq w_c & \forall g \in V,\ \forall c \in L & (5.19) \\
& \lambda_c \geq a_c^g & \forall g \in V,\ \forall c \in L & (5.20) \\
& \displaystyle\sum_{c \in L} a_c^g \leq x_g & \forall g \in V & (5.21) \\
& a_c^g = 0 & \forall c \in L,\ \forall g \notin c & (5.22) \\
& d_i = x_i * T_{key} & \forall i \in inputs(G) & (5.23) \\
& d_g \geq T_g + x_g * T_{key} + d_{g'} & \begin{array}{r} \forall g \in V \backslash inputs(G) \\ \forall g' \in \delta^-(g) \end{array} & (5.24) \\
& d_{max} \geq d_o & \forall o \in outputs(G) & (5.25) \\
& x_g \in \{0,1\} & \forall g \in V & (5.26) \\
& \lambda_c \in \{0,1\},\ w_c \in \{0,\dots,W_c\} & \forall c \in L & (5.27) \\
& a_c^g \in \{0,1\} & \forall c \in L,\ \forall g \in V & (5.28) \\
& d_{max} \geq 0,\ d_g \geq 0 & \forall g \in V & (5.29)
\end{cases}
$$

Le modèle M_5 représente le problème d'insertion de portes clés multi-objectifs. Dans la suite des travaux, nous avons priorisé l'intégration des algorithmes dans le flot de conception, en incluant les diverses stratégies de résolutions.

5.4 Extension de sécurité

Nous proposons une extension de sécurité s'insérant dans un flot de conception, entre la synthèse logique et le placement-routage. La Figure 5.5 image notre extension de sécurité (en vert) qui s'insère entre la synthèse logique et le placement-routage (en bleu). La synthèse logique est effectuée avec Yosys [Wol16] et le placement-routage est réalisé par Coriolis [Ale+05]. Dans notre extension, l'*ATPG* est calculé avec TetraMAX [Inc05]. La construction du graphe d'interférence complet est décrite dans la Section 4.2, la recherche du sous-graphe optimal (i.e., positions d'insertion optimales) est effectuée par l'Algorithme 4.

À l'exception de TetraMAX, les outils de conception utilisés sont *open-source*, mais il existe des alternatives *open-source* pour l'$ATPG$ comme Fault [AGS21]. Il est alors possible de construire un flot entièrement *open-source*.

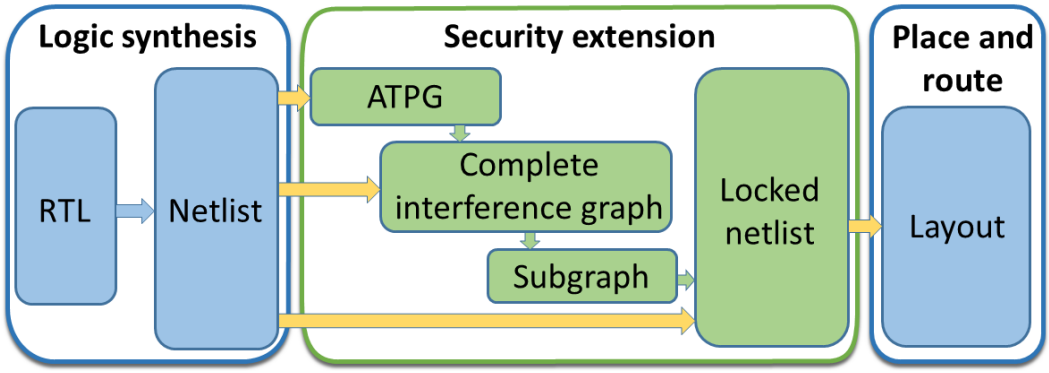


Figure 5.5. : Insertion de l'extension de sécurité dans un flot de conception

L'extension de sécurité est écrite en Julia 1.8 [Bez+17]. Tous les calculs ont été effectués sur un Windows 10 64 bits, Intel $i7-8665U$ avec : 4 cœurs, 8 threads, 16 Go de mémoire. Les *benchmarks* utilisés sont $ISCAS85$ [BF85] et $ITC99$ [Dav99].

Les tableaux 5.3 et 5.4 agrègent les résultats obtenus par notre outil. Ces tableaux possèdent 11 colonnes regroupées en 4 catégories :

— la description des circuits : le nom, le nombre de portes logiques et le nombre de portes clés à ajouter

Information du circuit			Pré-traitement		BB			BB+S		
Circuit	#portes	#portes clés	Temps GIC (s)	Temps cliques (s)	Temps (s)	Sécurité	Délai (%)	Temps (s)	Sécurité	Délai (%)
c17	8	2 (25%)	5.17	0.3	0.58	2^2	60.89	0.19	2^2	60.89
		4 (50%)			0.67	2^3	136.27	0.23	2^3	90.69
		6 (75%)			0.7	$2^{3.5849}$	136.75	0.25	$2^{3.5849}$	90.69
c432	134	7 (5%)	2.44	0.01	0.54	2^7	11.89	0.35	2^7	0
		13 (10%)			0.45	$2^{10.0112}$	11.89	0.04	$2^{7.3398}$	0
		20 (15%)			0.41	$2^{10.1799}$	17.83	0.35	$2^{7.6438}$	15.81
c499	207	10 (5%)	28.79	17.31	1.19	2^{10}	9.76	0.47	2^{10}	19.52
		21 (10%)			0.87	2^{21}	9.76	0.62	2^{21}	19.52
		31 (15%)			0.79	2^{31}	29.28	0.52	2^{31}	19.52
c880	228	11 (5%)	3.85	0.01	0.62	$2^{7.1699}$	0	0.53	$2^{7.1699}$	0
		23 (10%)			0.65	$2^{8.5999}$	0	0.75	$2^{8.5999}$	0
		34 (15%)			1.07	$2^{9.1699}$	4.33	0.69	$2^{9.1699}$	4.33
c1355	207	10 (5%)	20.18	18.98	1.05	2^{10}	9.76	0.61	2^{10}	17.3
		21 (10%)			1.18	2^{21}	29.29	0.64	2^{21}	19.52
		31 (15%)			1.22	2^{31}	29.29	0.58	2^{31}	19.52
c1908	173	9 (5%)	7.07	0.01	0.38	$2^{8.0112}$	13.01	0.38	$2^{6.1699}$	10.5
		17 (10%)			0.48	$2^{8.2479}$	15.5	0.49	$2^{7.0223}$	15.5
		26 (15%)			1.72	$2^{8.4093}$	21.6	1.41	$2^{7.2667}$	21.27
c2670	403	20 (5%)	148.55	0.09	1.2	2^{20}	5.33	0.83	2^{20}	0
		40 (10%)			1.17	2^{40}	2.08	0.87	2^{40}	2.08
		31 (15%)			1.04	2^{60}	12.74	0.85	2^{60}	2.08
c3540	491	25 (5%)	158.7	0.12	2.09	2^{25}	0	1.4	2^{25}	2.56
		49 (10%)			1.8	2^{49}	0	1.49	2^{49}	0.35
		74 (15%)			1.76	2^{74}	5.86	1.42	2^{74}	2.01
c5315	758	38 (5%)	339.02	0.12	1.92	2^{38}	5.14	1.54	2^{38}	2.27
		76 (10%)			2.83	2^{71}	5.14	2.85	2^{71}	5.14
		114 (15%)			3.93	2^{71}	10.48	2.79	2^{71}	5.14
c6288	1480	74 (5%)	616.65	0.23	5.3	$2^{27.0000}$	12.34	4.31	$2^{27.0000}$	5.63
		148 (10%)			6.82	$2^{27.0000}$	22.55	5.0	$2^{27.0000}$	10.1
		222 (15%)			7.2	$2^{27.0000}$	22.55	3.72	$2^{27.0000}$	13.78
c7552	949	47 (5%)	2804.82	0.9	3.41	2^{47}	5.92	4.96	2^{47}	7.09
		95 (10%)			3.43	2^{95}	5.92	3.49	2^{95}	7.09
		142 (15%)			4.96	2^{142}	17.88	4.41	2^{142}	19.96

Tableau 5.3. : Temps d'exécution et résultats de l'Algorithme 4 sans et avec la stratégie de délai sur les benchmarks $ISCAS85$

— pré-traitement : le temps de calcul du GIC (Section 4.2) et le temps de calcul de la liste de cliques maximales du GIC

— les résultats de l'Algorithme 4 : le temps de calcul, la sécurité obtenue et le surcoût en délai

— les résultats de l'Algorithme 4 avec la stratégie de résolution tenant compte du délai (Section 5.3.1) : le temps de calcul, la sécurité obtenue et le surcoût en délai

On observe que les algorithmes de résolution trouvent la solution optimale en moins de 25 secondes pour ces circuits. Le calcul de la liste de cliques maximales prend également moins de 20 secondes pour ces circuits. En agrégeant les temps de calcul, nous obtenons la Figure 5.6. Remarquons que sur les circuits de plus de 200

Information du circuit			Pré-traitement		BB			BB+S		
circuit	#portes	#portes clés	Temps GIC (s)	Temps cliques (s)	Temps (s)	Sécurité	Délai (%)	Temps (s)	Sécurité	Délai (%)
b01	20	5 (25%)	5.47	0.29	0.7	$2^{3.5849}$	45.28	0.31	$2^{3.5849}$	30.09
		10 (50%)			1.08	$2^{4.4594}$	85.97	0.31	$2^{4.4594}$	58.74
		15 (75%)			0.77	2^{5}	114.63	0.33	2^{5}	58.74
b02	16	4 (25%)	5.41	0.28	0.03	$2^{3.3219}$	95.7	0.08	$2^{3.0}$	33.79
		8 (50%)			0.06	$2^{4.1699}$	95.7	0.04	2^{4}	81.16
		12 (75%)			0.04	$2^{4.7004}$	127.61	0.04	$2^{4.5849}$	81.16
b03	63	16 (25%)	0.12	0.01	0.24	$2^{5.9068}$	95.18	0.24	$2^{5.2479}$	62.95
		32 (50%)			1.48	$2^{6.5235}$	96.67	0.13	$2^{6.1292}$	62.95
		47 (75%)			1.08	$2^{6.9307}$	117.98	0.14	$2^{6.6438}$	127.4
b04	288	14 (5%)	8.42	0.01	0.74	2^{14}	12.83	0.86	2^{14}	5.93
		29 (10%)			0.71	$2^{23.0000}$	19.24	0.9	$2^{20.0001}$	5.93
		43 (15%)			0.93	$2^{23.0000}$	19.24	0.92	$2^{20.0001}$	12.73
b05	317	16 (5%)	9.4	0.01	0.96	2^{16}	15.97	1.19	$2^{14.0003}$	1.91
		32 (10%)			0.9	$2^{17.0225}$	18.35	1.21	$2^{14.1718}$	7.97
		48 (15%)			11.17	$2^{17.0231}$	31.53	7.91	$2^{14.1758}$	14.73
b06	24	6 (5%)	1.95	0.01	0.06	$2^{3.5849}$	92.04	0.09	$2^{3.5849}$	52.23
		12 (10%)			0.09	$2^{4.5849}$	114.79	0.07	$2^{4.5849}$	74.98
		48 (15%)			0.07	$2^{5.1699}$	119.41	0.07	$2^{5.1699}$	119.41
b07	214	11 (5%)	2.27	0.01	0.78	$2^{6.0443}$	12.9	0.79	$2^{5.5849}$	4.13
		21 (10%)			0.68	$2^{6.5235}$	17.03	1.87	$2^{6.2094}$	16.26
		32 (15%)			0.99	$2^{6.9307}$	17.03	0.72	$2^{6.7004}$	22.82
b08	98	5 (5%)	1.12	0.01	0.26	2^{5}	9.94	0.29	2^{5}	5.64
		10 (10%)			0.21	2^{10}	25.51	0.28	2^{10}	15.58
		15 (15%)			0.38	2^{15}	15.58	0.27	2^{15}	25.51
b09	58	3 (5%)	0.25	0.01	0.15	$2^{2.5849}$	15.42	0.2	$2^{2.5849}$	15.42
		6 (10%)			0.1	$2^{3.5849}$	15.42	0.15	$2^{3.5849}$	15.42
		9 (15%)			0.32	$2^{4.1699}$	15.42	0.14	$2^{4.1699}$	15.42
b10	123	6 (5%)	0.81	0.01	0.43	$2^{4.3219}$	0.0	0.79	$2^{4.3219}$	0.0
		12 (10%)			0.33	$2^{5.5849}$	17.8	0.45	$2^{5.5849}$	17.8
		18 (15%)			1.01	2^{6}	17.8	0.47	2^{6}	17.8
b11	304	15 (5%)	13.73	0.02	1.07	2^{15}	6.68	1.14	2^{15}	1.69
		30 (10%)			0.8	$2^{28.0000}$	20.04	1.03	$2^{25.0000}$	10.94
		46 (15%)			2.07	$2^{28.0000}$	20.04	1.0	$2^{25.0000}$	10.94
b12	512	26 (5%)	47.48	0.02	1.63	$2^{17.0056}$	19.94	1.37	$2^{17.0056}$	1.59
		51 (10%)			1.26	$2^{18.0000}$	25.28	19.2	$2^{17.6439}$	35.26
		77 (15%)			19.14	$2^{18.0005}$	49.86	11.65	$2^{17.6446}$	35.26
b13	187	9 (5%)	1.36	0.01	1.49	$2^{4.8073}$	14.17	0.4	$2^{4.5849}$	0.0
		19 (10%)			1.15	$2^{5.8073}$	14.17	0.4	$2^{5.6438}$	0.1
		28 (15%)			0.53	$2^{6.3219}$	14.17	0.4	$2^{6.2094}$	0.1
b14	2534	127 (5%)	23 192	1.14	9.31	2^{127}	0.0	8.05	2^{127}	0.0
		253 (10%)			7.71	2^{253}	4.68	8.44	2^{253}	7.74
		380 (15%)			9.51	2^{380}	17.43	8.21	2^{380}	11.69
b15	3873	194 (5%)	56 644	2.22	16.37	2^{194}	0.92	12.27	2^{194}	2.12
		387 (10%)			23.16	2^{276}	7.14	23.11	2^{276}	7.14
		581 (15%)			18.54	2^{276}	10.49	16.68	2^{276}	8.12

Tableau 5.4. : Temps d'exécution et résultats de l'Algorithme 4 sans et avec la stratégie de délai sur les benchmarks $ITC99$

sommets, le calcul du GIC prend en moyenne 87% du temps de calcul. Calculer le GIC est polynomial contrairement aux calculs de la liste de cliques maximales et à la résolution du problème. Néanmoins, c'est le calcul le plus long de notre outil.

Sans ce calcul, nous ne pouvons pas tester l'outil sur des circuits plus grands. C'est le point limitant de nos expérimentations.

Circuit	Algorithme 4				*SLL* [Yas+16b]			
	sécurité	délai (%)	surface (%)	calcul (s)	sécurité	délai (%)	surface (%)	calcul (s)
c5315	2^{15}	0	2	339.41	$\approx 2^{14}$	0	2	≈ 2200
	2^{14}	0	1.85	339.37				
c7552	2^{200}	18%	21	2805.51	$\approx 2^{128}$	0	21	≈ 15000
	2^{128}	13.59%	13.48	2805.59				

Tableau 5.5. : Comparaison de résultats avec le SLL

Notre méthode est une extension du SLL [Yas+16b]. Nous avons deux circuits en commun : $c5315$ et $c7552$. Nous pouvons comparer nos résultats. Le Tableau 5.5 compare les résultats en termes de sécurité et de délai obtenus avec un surcoût en surface fixé pour l'Algorithme 4 (BB) et le meilleur algorithme, nommé $SLJI$, dans [Yas+16b]. Les données sont extraites des graphiques publiés. Nous avons imposé l'augmentation de surface, que nous avons extraite des données, pour notre algorithme. Puis nous avons imposé une sécurité équivalente pour observer la surface obtenue. Nous constatons que notre algorithme calcule plus rapidement un résultat, étant 10 fois plus rapide pour le circuit $c5315$ et 5 fois plus rapide pour le $c7552$. Dans le cas de notre outil, la majeure partie de ce temps est due au calcul du GIC. Le surcoût en délai obtenu est supérieur pour notre outil dans un cas, celui du circuit $c7552$. En effet, il possède un grand nombre de solutions optimales, or notre outil n'en choisit qu'une et la contrainte temporelle n'est pas suffisante dans ce cas précis. Dans ce cas, il a sélectionné une solution avec un fort surcoût.

Par définition, notre outil trouve la sécurité optimale pour une surface donnée. Cela signifie qu'à surface équivalente, il apporte donc plus de sécurité que le SLL. De plus, on observe sécurité équivalente, notre outil utilise moins de surface. Enfin, l'outil est intégré dans le flot de conception et l'exécution des algorithmes est plus rapide.

5.5 Conclusion

Dans ce chapitre, nous avons proposé deux modélisations temporelles avec une granularité différente. La première est la plus fine (modèle M_3) et requiert beaucoup d'informations, ainsi qu'une intégration avec un outil de placement-routage. Travaillant sur des technologies de gravure de plus de $100nm$, nous pouvons négliger le délai des fils. En conséquence, le deuxième modèle est plus granulaire (modèle M_4),

mais suffisant pour notre cas d'usage. Ce modèle temporel cherche à ajouter des portes clés dans le circuit, en minimisant l'impact sur le délai, indépendamment de la sécurité. Les résultats numériques montrent que les niveaux de sécurité atteints par le modèle M_4 sont faibles comparés à ceux obtenus en optimisant la sécurité. Nous avons alors proposé d'intégrer une contrainte temporelle dans le modèle de sécurité. Les résultats montrent que cette contrainte temporelle n'impacte pas la sécurité dans plus de la moitié des cas étudiés, et réduit le surcoût en délai. Cette stratégie de résolution permet d'obtenir, sans surcoût de calcul, une sécurité intéressante, tout en diminuant le délai du circuit. Nous avons de plus proposé une modélisation multi-objectifs du problème, dont la résolution constitue un axe de poursuite pour ces travaux.

Nous avons ensuite proposé une extension du flot de conception pour intégrer notre mesures de sécurité, et pour montrer la faisabilité d'un tel outil. La mise en place de cette extension a permis de tester les algorithmes sur un plus grand nombre de circuits, démontrant ainsi sa généralité. Nous avons également remarqué une limite de l'outil, liée à un algorithme de pré-traitement. Cet algorithme étant sommaire, il est assurément possible de proposer des optimisations algorithmiques afin d'accélérer les temps de calcul.

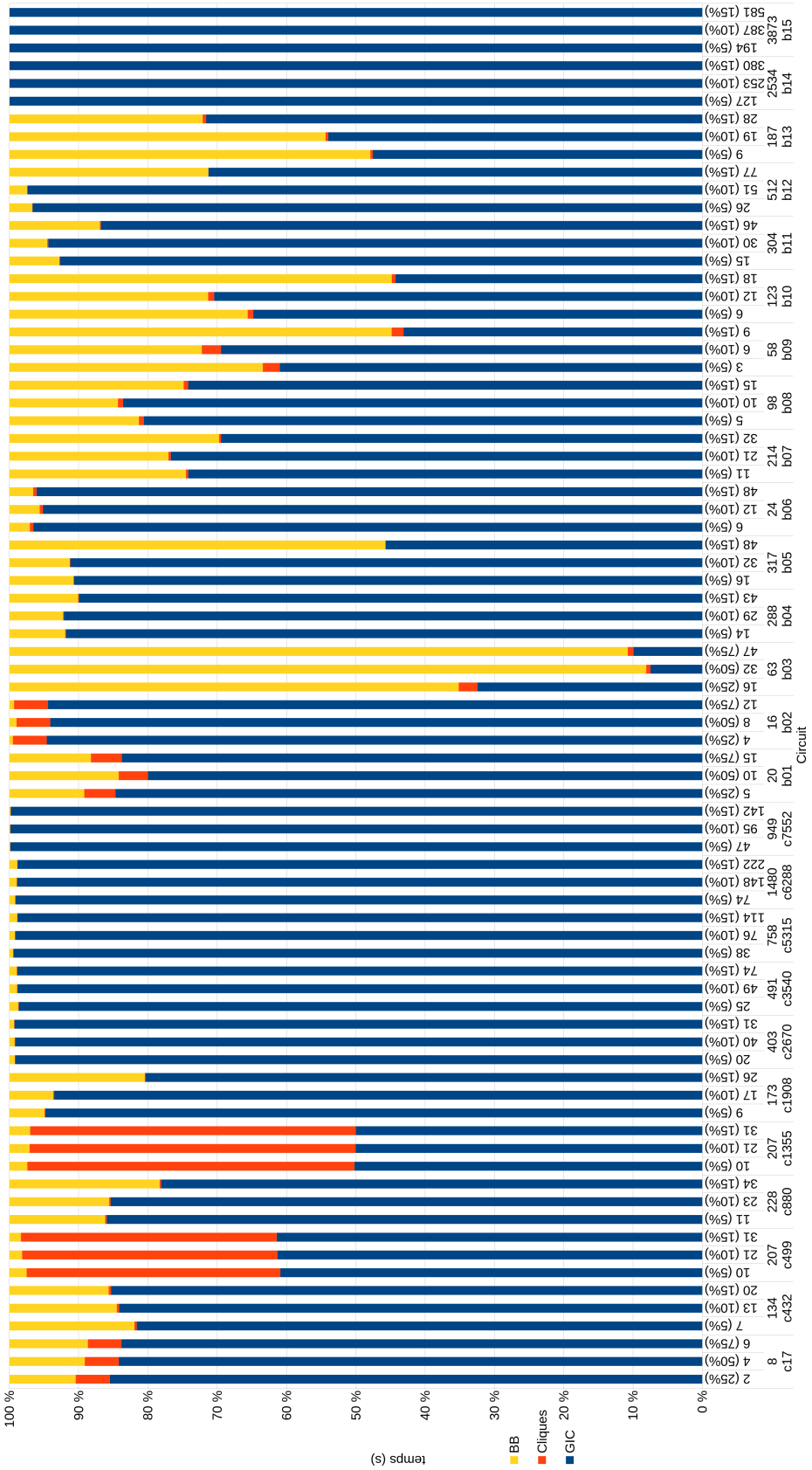


Figure 5.6. : Répartition du temps de calcul entre la création du GIC, le calcul de la liste des cliques maximales et l'Algorithme 4

6 Conclusions et Perspectives

6.1 Conclusions

L'externalisation de la production de circuits a engendré de nouvelles problématiques de sécurité, telles que les Chevaux de Troie Matériels (CTM), la surproduction, le vol d'IP et la contrefaçon. Le *logic locking* est une méthode visant à prévenir ces menaces. La complexité de mise en œuvre de cette méthode nécessite des outils automatiques pour leur large diffusion. Cette automatisation requiert une modélisation mathématique et des algorithmes d'optimisation pour maximiser la sécurité tout en limitant les contreparties.

Parmi les différentes propositions de *logic locking*, nous avons choisi le *Strong Logic Locking* (SLL). Cette méthode est souvent hybridée avec les approches $post\text{-}SAT$. Elle est donc pertinente pour le domaine, mais sa résolution est traditionnellement effectuée par des algorithmes gloutons, sans garantie de résultats optimaux.

Nous avons alors proposé de la modéliser mathématiquement par des formulation de programmation linéaire en nombre entier et des graphes. Pour ce faire, nous avons d'abord modifié l'approche de résolution. Puis, nous avons étendu la mesure de sécurité.

Ces modifications effectuées, nous avons divisé notre problème de sécurité en deux étapes : un pré-traitement qui calcule le Graphe d'Interference Complet (GIC) avec un algorithme polynomial, puis le calcul de la partition en cliques pondérées maximales qui correspond aux positions d'insertion des portes clés dans le circuit. Cette résolution s'effectue avec un algorithme de *Branch & Bound* dédié. Cet algorithme est exponentiel en la taille du GIC, qui dépend de la taille du circuit.

Nous avons prouvé que cette étape est $NP\text{-}DUR$ et $NON\text{-}APX$. Nous ne pouvons donc pas la résoudre en temps polynomial, ni même trouver une solution approximative avec un ratio constant en temps polynomial.

L'ajout de cette sécurité augmente le délai du circuit. Nous cherchons à modéliser le problème d'insertion de portes logiques dans le circuit en minimisant l'impact sur

le délai. Les résultats montrent un impact nul sur le délai du circuit, néanmoins, la sécurité obtenue est très inférieure aux résultats initiaux.

Nous avons alors proposé d'ajouter une contrainte de délai dans le modèle de sécurité. Cette contrainte montre des améliorations sur l'impact du délai, mais diminue partiellement la sécurité obtenue.

Cette diversité de solutions souligne la nécessité d'algorithmes d'optimisation multi-objectifs. Nous avons proposé une modélisation multi-objective du problème considérant la sécurité et le délai.

Une fois les travaux d'optimisation effectués, nous avons proposé une extension de sécurité pour le flot de conception. Cette extension s'appuie sur les algorithmes développés pendant la thèse. Elle nous a permis d'obtenir des résultats sur des circuits, démontrant ainsi la généralité de l'outil, son efficacité, mais également ses limites en termes de temps de calcul.

6.2 Perspectives

Notre thèse présente des résultats variés en mono-objectif selon l'objectif de sécurité ou de délai. Nous supposons l'existence de plusieurs solutions variées sur le front de Pareto du problème multi-objectifs. Une perspectives de cette thèse serait d'implémenter des algorithmes de résolution multi-objectifs comme une résolution exacte avec un algorithme de $Kirlik - Sayin$ [KS14], ou des approches heuristiques avec un algorithme génétique [FF98].

La plus grande limitation de notre outil est le temps de calcul du GIC. Les limites de cet algorithme ont été révélées en fin de thèse, lors des dernières expérimentations. Cet algorithme effectue une simulation du circuit pour certaines valeurs en prenant en compte des signaux inconnus. Nous sommes convaincus que des optimisations algorithmiques peuvent être effectuées.

Dans les résultats de l'outil, nous observons que la sécurité maximale n'est pas corrélée à la taille du circuit. Nous concluons que certains circuits sont plus sécurisables que d'autres. Ce constat pourrait être exploité par un concepteur cherchant à mesurer la sécurisabilité d'un circuit afin de concevoir un circuit plus sécurisable qu'un autre.

Publications

[Fon+23] Jonathan FONTAINE, Mohamed BENAZOUZ, Lilia ZAOURAR et Roselyne CHOTIN. “Improving Integrated Circuit Security using Mathematical Model Based on Clique Covering Reformulation”. In : *9th International Conference on Control, Decision and Information Technologies (CoDiT 2023)*. 2023, p. 1717-1722 (cf. p. 76).

[Fon+22] Jonathan FONTAINE, Lilia ZAOURAR, Mohamed BENAZOUZ et Roselyne CHOTIN. “Recherche de cliques pour un problème de cybersécurité matériel”. In : *23ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d’Aide à la Décision*. 2022.

[FZC21a] Jonathan FONTAINE, Lilia ZAOURAR et Roselyne CHOTIN. “Mathematical modelling of Logic Locking against the insertion of Hardware Trojan in an Intregated Circuit”. In : *31 European conference on operational research*. 2021.

[FZC21b] Jonathan FONTAINE, Lilia ZAOURAR et Roselyne CHOTIN. “Optimisation de contre-mesure à l’insertion de Hardware Trojan”. In : *22ème Conférence ROADEF de la société Française de Recherche Opérationnelle et Aide à la Décision*. 2021.

Bibliographie

[AGS21] Manar ABDELATTY, Mohamed GABER et Mohamed SHALAN. “Fault : Open-Source EDA’s Missing DFT Toolchain”. In : *IEEE Design & Test* 38.2 (2021), p. 45-52 (cf. p. 87).

[Agr+07] Dakshi AGRAWAL, Selcuk BAKTIR, Deniz KARAKOYUNLU, Pankaj ROHATGI et Berk SUNAR. “Trojan Detection using IC Fingerprinting”. In : *2007 IEEE Symposium on Security and Privacy (SP ’07)*. 2007, p. 296-310 (cf. p. 5).

[Ale+05] C. ALEXANDRE, H. CLEMENT, J.-P. CHAPUT et al. “TSUNAMI : an integrated timing-driven place and route research platform”. In : *Design, Automation and Test in Europe*. 2005, 920-921 Vol. 2 (cf. p. 87).

[BZ22] Johanna BAEHR et Alexander ZEH. “Post-Quantum Logic Locking”. In : (juin 2022) (cf. p. 39).

[BH09] Mainak BANGA et Michael S. HSIAO. “VITAMIN : Voltage inversion technique to ascertain malicious insertions in ICs”. In : *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2009, p. 104-107 (cf. p. 6).

[BFS16] Chongxi BAO, Domenic FORTE et Ankur SRIVASTAVA. “On Reverse Engineering-Based Hardware Trojan Detection”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.1 (2016), p. 49-57 (cf. p. 4).

[BB48] J. BARDEEN et W. H. BRATTAIN. “The Transistor, A Semi-Conductor Triode”. In : *Phys. Rev.* 74 (2 juill. 1948), p. 230-231 (cf. p. 1).

[BTZ10] Alex BAUMGARTEN, Akhilesh TYAGI et Joseph ZAMBRENO. “Preventing IC piracy using reconfigurable logic barriers”. In : *IEEE design & Test of computers* 27.1 (2010), p. 66-75 (cf. p. 28).

[Ber85] Claude BERGE. *Graphs and hypergraphs*. Elsevier Science Ltd., 1985 (cf. p. 11).

[Bez+17] Jeff BEZANSON, Alan EDELMAN, Stefan KARPINSKI et Viral B SHAH. “Julia : A fresh approach to numerical computing”. In : *SIAM Review* 59.1 (2017), p. 65-98 (cf. p. 63, 72, 82, 87).

[BF85] F BRGLEZ et H FUJIWARA. “Special Session on ATPG (Also introducing’A Neutral Netlist of 10 Combinational Benchmark Circuits’)”. In : *Int. Symp. On Circuits and Systems*. 1985 (cf. p. 63, 72, 82, 87).

[CB09] Rajat Subhra CHAKRABORTY et Swarup BHUNIA. “Security through obscurity : An approach for protecting Register Transfer Level hardware IP”. In : *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2009, p. 96-99 (cf. p. 7).

[CB10] Rajat Subhra CHAKRABORTY et Swarup BHUNIA. “RTL Hardware IP Protection Using Key-Based Control and Data Flow Obfuscation”. In : *2010 23rd International Conference on VLSI Design*. 2010, p. 405-410 (cf. p. 7).

[Cha+09] Rajat Subhra CHAKRABORTY, Francis WOLFF, Somnath PAUL, Christos PAPACHRISTOU et Swarup BHUNIA. “MERO : A Statistical Approach for Hardware Trojan Detection”. In : *Cryptographic Hardware and Embedded Systems - CHES 2009*. Sous la dir. de Christophe CLAVIER et Kris GAJ. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 396-410 (cf. p. 5).

[Cla06] Peter CLARK. “Fake NEC company found, says report”. In : *EETimes* (2006) (cf. p. 4).

[Coo71] Stephen A. COOK. “The Complexity of Theorem-Proving Procedures”. In : *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC ’71. New York, NY, USA : Association for Computing Machinery, 1971, 151–158 (cf. p. 18).

[CGM18] Maxime COZZI, Jean-Marc GALLIERE et Philippe MAURINE. “Exploiting Phase Information in Thermal Scans for Stealthy Trojan Detection”. In : *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018, p. 573-576 (cf. p. 5).

[Dav99] Scott DAVIDSON. “Characteristics of the ITC’99 benchmark circuits”. In : *IEEE International Test Synthesis Workshop (ITSW)*. 1999 (cf. p. 87).

[Dij59] E. W. DIJKSTRA. “A note on two problems in connexion with graphs”. In : *Numerische Mathematik* 1.1 (déc. 1959), 269–271 (cf. p. 18).

[Dup+14a] Sophie DUPUIS, Papa-Sidi BA, Giorgio DI NATALE, Marie-Lise FLOTTES et Bruno ROUZEYRE. “A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans”. In : *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. 2014, p. 49-54 (cf. p. 7).

[Dup+14b] Sophie DUPUIS, Papa-Sidi BA, Giorgio DI NATALE, Marie-Lise FLOTTES et Bruno ROUZEYRE. “A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans”. In : *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. IEEE. 2014, p. 49-54 (cf. p. 25, 29).

[DF19] Sophie DUPUIS et Marie-Lise FLOTTES. “Logic Locking : A Survey of Proposed Methods and Evaluation Metrics”. In : *Journal of Electronic Testing* 35 (juin 2019), p. 1-19 (cf. p. 25).

[ELS10] David EPPSTEIN, Maarten LÖFFLER et Darren STRASH. “Listing all maximal cliques in sparse graphs in near-optimal time”. In : *Algorithms and Computation : 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I 21*. Springer. 2010, p. 403-414 (cf. p. 65).

[FF98] C.M. FONSECA et P.J. FLEMING. “Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation”. In : *IEEE Transactions on Systems, Man, and Cybernetics - Part A : Systems and Humans* 28.1 (1998), p. 26-37 (cf. p. 94).

[Fon+23] Jonathan FONTAINE, Mohamed BENAZOUZ, Lilia ZAOURAR et Roselyne CHOTIN. “Improving Integrated Circuit Security using Mathematical Model Based on Clique Covering Reformulation”. In : *9th International Conference on Control, Decision and Information Technologies (CoDiT 2023)*. 2023, p. 1717-1722 (cf. p. 76).

[Fon+22] Jonathan FONTAINE, Lilia ZAOURAR, Mohamed BENAZOUZ et Roselyne CHOTIN. “Recherche de cliques pour un problème de cybersécurité matériel”. In : *23ème édition du congrès annuel de la Société Française de Recherche Opérationnelle et d’Aide à la Décision*. 2022.

[FZC21a] Jonathan FONTAINE, Lilia ZAOURAR et Roselyne CHOTIN. “Mathematical modelling of Logic Locking against the insertion of Hardware Trojan in an Intregated Circuit”. In : *31 European conference on operational research*. 2021.

[FZC21b] Jonathan FONTAINE, Lilia ZAOURAR et Roselyne CHOTIN. “Optimisation de contre-mesure à l’insertion de Hardware Trojan”. In : *22ème Conférence ROADEF de la société Française de Recherche Opérationnelle et Aide à la Décision*. 2021.

[GJ79] Michael R GAREY et David S JOHNSON. “Computers and intractability”. In : (1979) (cf. p. 18).

[Gub+23] Kevin Immanuel GUBBI, Banafsheh SABER LATIBARI, Anirudh SRIKANTH et al. “Hardware Trojan Detection Using Machine Learning : A Tutorial”. In : *ACM Trans. Embed. Comput. Syst.* 22.3 (avr. 2023) (cf. p. 6).

[Gur23] GUROBI OPTIMIZATION, LLC. *Gurobi Optimizer Reference Manual*. 2023 (cf. p. 63, 66, 68, 72, 82).

[HO13] Jie HAN et Michael ORSHANSKY. “Approximate computing : An emerging paradigm for energy-efficient design”. In : *2013 18th IEEE European Test Symposium (ETS)*. 2013, p. 1-6 (cf. p. 28).

[Hos+17] Fakir Sharif HOSSAIN, Tomokazu YONEDA, Michiko INOUE et Alex ORAILOGLU. “Detecting hardware Trojans without a Golden IC through clock-tree defined circuit partitions”. In : *2017 22nd IEEE European Test Symposium (ETS)*. 2017, p. 1-6 (cf. p. 7).

[Inc05] Synopsys INC. *TetraMAX ATPG user guide*. 2005 (cf. p. 87).

[JM08] Yier JIN et Yiorgos MAKRIS. “Hardware Trojan detection using path delay fingerprint”. In : *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2008, p. 51-57 (cf. p. 5).

[Kam+20] Hadi M. KAMALI, Kimia Z. AZAR, Houman HOMAYOUN et Avesta SASAN. “On Designing Secure and Robust Scan Chain for Protecting Obfuscated Logic”. In : *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. GLSVLSI ’20. New York, NY, USA : Association for Computing Machinery, 2020, 217–222 (cf. p. 40, 42).

[Kam+22] Hadi Mardani KAMALI, Kimia Zamiri AZAR, Farimah FARAHMANDI et Mark TEHRANIPOOR. *Advances in Logic Locking : Past, Present, and Prospects*. Cryptology ePrint Archive, Paper 2022/260. https://eprint.iacr.org/2022/260. 2022 (cf. p. 41).

[Kar75] Richard M KARP. "On the computational complexity of combinatorial problems". In : *Networks* 5.1 (1975), p. 45-68 (cf. p. 18, 54).

[Kar+10] Ramesh KARRI, Jeyavijayan RAJENDRAN, Kurt ROSENFELD et Mohammad TEHRANIPOOR. "Trustworthy Hardware : Identifying and Classifying Hardware Trojans". In : *Computer* 43.10 (2010), p. 39-46 (cf. p. 4).

[KLS00] Sanjeev KHANNA, Nathan LINIAL et Shmuel SAFRA. "On the hardness of approximating the chromatic number". In : *Combinatorica* 20.3 (2000), p. 393-415 (cf. p. 55).

[KS14] Gokhan KIRLIK et Serpil SAYIN. "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems". In : *European Journal of Operational Research* 232.3 (2014), p. 479-488 (cf. p. 94).

[Kne+20] Johann KNECHTEL, Elif Bilge KAVUN, Francesco REGAZZONI et al. "Towards Secure Composition of Integrated Circuits and Electronic Systems : On the Role of EDA". In : *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, p. 508-513 (cf. p. 42).

[LT15] Yu-Wei LEE et Nur A. TOUBA. "Improving logic obfuscation via logic cone analysis". In : *2015 16th Latin-American Test Symposium (LATS)*. 2015, p. 1-6 (cf. p. 29).

[Lin+23] Shang-Wei LIN, Si-Han CHEN, Tzu-Fan WANG et Yean-Ru CHEN. *A Quantum SMT Solver for Bit-Vector Theory*. 2023. arXiv : 2303.09353 [cs.LO] (cf. p. 39).

[Liu+20] Yuntao LIU, Michael ZUZAK, Yang XIE, Abhishek CHAKRABORTY et Ankur SRIVASTAVA. "Strong Anti-SAT : Secure and Effective Logic Locking". In : *2020 21st International Symposium on Quality Electronic Design (ISQED)*. 2020, p. 199-205 (cf. p. 37, 38).

[Mar+17] Andrea MARCELLI, Marco RESTIFO, Ernesto SANCHEZ et Giovanni SQUILLERO. "An evolutionary approach to hardware encryption and Trojan-horse mitigation". In : *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, p. 1593-1598 (cf. p. 29).

[Mas+17] Mohamed El MASSAD, Jun ZHANG, Siddharth GARG et Mahesh V. TRIPUNITARA. *Logic Locking for Secure Outsourced Chip Fabrication : A New Attack and Provably Secure Defense Mechanism*. 2017. arXiv : 1703.10187 [cs.CR] (cf. p. 40, 41).

[MM65] John W MOON et Leo MOSER. "On cliques in graphs". In : *Israel journal of Mathematics* 3.1 (1965), p. 23-28 (cf. p. 65).

[Nar+10] Seetharam NARASIMHAN, Dongdong DU, Rajat Subhra CHAKRABORTY et al. "Multiple-parameter side-channel analysis : A non-invasive hardware Trojan detection approach". In : *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2010, p. 13-18 (cf. p. 5).

[NHB15] Arash NEJAT, David HELY et Vincent BEROULLE. “Facilitating side channel analysis by obfuscation for Hardware Trojan detection”. In : *2015 10th International Design & Test Symposium (IDT)*. 2015, p. 129-134 (cf. p. 30).

[NHB16] Arash NEJAT, David HELY et Vincent BEROULLE. “How logic masking can improve path delay analysis for Hardware Trojan detection”. In : *2016 IEEE 34th International Conference on Computer Design (ICCD)*. 2016, p. 424-427 (cf. p. 30).

[NHB18] Arash NEJAT, David HELY et Vincent BEROULLE. “ESCALATION : Leveraging logic masking to facilitate path-delay-based hardware Trojan detection methods”. In : *Journal of Hardware and Systems Security* 2 (2018), p. 83-96 (cf. p. 30).

[Nej+19] Arash NEJAT, Zahra KAZEMI, Vincent BEROULLE, David HELY et Mahdi FAZELI. “Restricting Switching Activity Using Logic Locking to Improve Power Analysis-Based Trojan Detection”. In : *2019 IEEE 4th International Verification and Security Workshop (IVSW)*. 2019, p. 49-54 (cf. p. 30).

[Ngo+15] X-T. NGO, I. EXURVILLE, S. BHASIN et al. “Hardware Trojan detection by delay and electromagnetic measurements”. In : *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2015, p. 782-787 (cf. p. 5).

[PMX98] Panos M PARDALOS, Thelma MAVRIDOU et Jue XUE. “The graph coloring problem : A bibliographic survey”. In : *Handbook of Combinatorial Optimization : Volume1–3* (1998), p. 1077-1141 (cf. p. 55).

[PM15] Stephen M. PLAZA et Igor L. MARKOV. “Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.6 (2015), p. 961-971 (cf. p. 28).

[PT13] Ayoush Johari PRACHI TIWARI Anupama. “Study and Comparison of Open Source and Licensed VLSI CAD Tools using CMOS Design Methodology”. In : (2013) (cf. p. 3).

[RSK14a] J. RAJENDRAN, O. SINANOGLU et R. KARRI. “Regaining Trust in VLSI Design : Design-for-Trust Techniques”. In : *Proceedings of the IEEE* 102.8 (août 2014), p. 1266-1282 (cf. p. 4).

[Raj+12a] Jeyavijayan RAJENDRAN, Youngok PINO, Ozgur SINANOGLU et Ramesh KARRI. “Logic encryption : A fault analysis perspective”. In : *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, p. 953-958 (cf. p. 7, 26, 28, 29).

[Raj+12b] Jeyavijayan RAJENDRAN, Youngok PINO, Ozgur SINANOGLU et Ramesh KARRI. “Security Analysis of Logic Obfuscation”. In : *Proceedings of the 49th Annual Design Automation Conference*. DAC ’12. New York, NY, USA : Association for Computing Machinery, 2012, 83–89 (cf. p. 28, 40, 43, 44).

[Raj+15] Jeyavijayan RAJENDRAN, Huan ZHANG, Chi ZHANG et al. “Fault Analysis-Based Logic Encryption”. In : *IEEE Transactions on Computers* 64.2 (2015), p. 410-424 (cf. p. 28).

[RSK14b] Michael RATHMAIR, Florian SCHUPFER et Christian KRIEG. “Applied formal methods for hardware Trojan detection”. In : *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2014, p. 169-172 (cf. p. 5).

[RBS67] J Paul ROTH, Willard G BOURICIUS et Peter R SCHNEIDER. “Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits”. In : *IEEE Transactions on Electronic Computers* 5 (1967), p. 567-580 (cf. p. 50).

[RKM08] Jarrod A. ROY, Farinaz KOUSHANFAR et Igor L. MARKOV. “EPIC : Ending Piracy of Integrated Circuits”. In : *2008 Design, Automation and Test in Europe*. 2008, p. 1069-1074 (cf. p. 7, 26).

[STP10] Hassan SALMANI, Mohammad TEHRANIPOOR et Jim PLUSQUELLIC. “A layout-aware approach for improving localized switching to detect hardware Trojans in integrated circuits”. In : *2010 IEEE International Workshop on Information Forensics and Security*. 2010, p. 1-6 (cf. p. 7).

[Sam+16] Mohammad Saleh SAMIMI, Ehsan AERABI, Zahra KAZEMI, Mahdi FAZELI et Ahmad PATOOGHY. “Hardware enlightening : No where to hide your Hardware Trojans !” In : *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2016, p. 251-256 (cf. p. 29).

[Sen+18] Abhrajit SENGUPTA, Mohammed NABEEL, Muhammad YASIN et Ozgur SINANOGLU. “ATPG-based cost-effective, secure logic locking”. In : *2018 IEEE 36th VLSI Test Symposium (VTS)*. 2018, p. 1-6 (cf. p. 37).

[Set09] Burr SETTLES. “Active learning literature survey”. In : (2009) (cf. p. 34).

[Sha+19] Bicky SHAKYA, Xiaolin XU, Mark TEHRANIPOOR et Domenic FORTE. “CAS-Lock : A Security-Corruptibility Trade-off Resilient Logic Locking Scheme”. In : *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (2019), 175–202 (cf. p. 38, 39).

[Sha+17a] Kaveh SHAMSI, Meng LI, Travis MEADE et al. “AppSAT : Approximately deobfuscating integrated circuits”. In : *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2017, p. 95-100 (cf. p. 34).

[Sha+17b] Kaveh SHAMSI, Meng LI, Travis MEADE et al. “Cyclic Obfuscation for Creating SAT-Unresolvable Circuits”. In : *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. New York, NY, USA : Association for Computing Machinery, 2017, 173–178 (cf. p. 35).

[SS15] Seyed Mohammad Hossein SHEKARIAN et Morteza SAHEB ZAMANI. “Improving hardware Trojan detection by retiming”. In : *Microprocessors and Microsystems* 39.3 (2015), p. 145-156 (cf. p. 30).

[SZ17] Yuanqi SHEN et Hai ZHOU. “Double DIP : Re-Evaluating Security of Logic Encryption Algorithms”. In : *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. New York, NY, USA : Association for Computing Machinery, 2017, 179–184 (cf. p. 35).

[SS19] Deepak SIRONE et Pramod SUBRAMANYAN. “Functional Analysis Attacks on Logic Locking”. In : *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, p. 936-939 (cf. p. 37).

[SRM15] Pramod SUBRAMANYAN, Sayak RAY et Sharad MALIK. “Evaluating the security of logic encryption algorithms”. In : *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2015, p. 137-143 (cf. p. 30, 31).

[SD07] G. Edward SUH et Srinivas DEVADAS. “Physical Unclonable Functions for Device Authentication and Secret Key Generation”. In : *2007 44th ACM/IEEE Design Automation Conference*. 2007, p. 9-14 (cf. p. 26).

[Syn] SYNOPSYS. *EDA Forms the Basis for Designing Secure Systems* (cf. p. 42).

[Syn23] SYNOPSYS. *Key Requirements for Automotive SoC Design*. 2023 (cf. p. 42).

[TK10] Mohammad TEHRANIPOOR et Farinaz KOUSHANFAR. “A Survey of Hardware Trojan Taxonomy and Detection”. In : *IEEE Design & Test of Computers* 27.1 (2010), p. 10-25 (cf. p. 6).

[TJ09] Randy TORRANCE et Dick JAMES. “The State-of-the-Art in IC Reverse Engineering”. In : *Cryptographic Hardware and Embedded Systems - CHES 2009*. Sous la dir. de Christophe CLAVIER et Kris GAJ. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 363-381 (cf. p. 30).

[Tuy+06] Pim TUYLS, Geert-Jan SCHRIJEN, Boris ŠKORIĆ et al. “Read-Proof Hardware from Protective Coatings”. In : *Cryptographic Hardware and Embedded Systems - CHES 2006*. Sous la dir. de Louis GOUBIN et Mitsuru MATSUI. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, p. 369-383 (cf. p. 36).

[Wol16] Clifford WOLF. “Yosys open synthesis suite”. In : (2016) (cf. p. 87).

[Wol+08] Francis WOLFF, Chris PAPACHRISTOU, Swarup BHUNIA et Rajat S. CHAKRABORTY. “Towards Trojan-Free Trusted ICs : Problem Analysis and Detection Scheme”. In : *2008 Design, Automation and Test in Europe*. 2008, p. 1362-1365 (cf. p. 5).

[XFT14] Kan XIAO, Domenic FORTE et Mohammed TEHRANIPOOR. “A Novel Built-In Self-Authentication Technique to Prevent Inserting Hardware Trojans”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (2014), p. 1778-1791 (cf. p. 7).

[XS16] Yang XIE et Ankur SRIVASTAVA. “Mitigating SAT Attack on Logic Locking”. In : *Cryptographic Hardware and Embedded Systems – CHES 2016*. Sous la dir. de Benedikt GIERLICHS et Axel Y. POSCHMANN. Berlin, Heidelberg : Springer Berlin Heidelberg, 2016, p. 127-146 (cf. p. 33).

[XS19] Yang XIE et Ankur SRIVASTAVA. “Anti-SAT : Mitigating SAT Attack on Logic Locking”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.2 (2019), p. 199-207 (cf. p. 33, 34).

[Yas+16a] Muhammad YASIN, Bodhisatwa MAZUMDAR, Jeyavijayan J V RAJENDRAN et Ozgur SINANOGLU. “SARLock : SAT attack resistant logic locking”. In : *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2016, p. 236-241 (cf. p. 32).

[Yas+16b] Muhammad YASIN, Jeyavijayan JV RAJENDRAN, Ozgur SINANOGLU et Ramesh KARRI. “On Improving the Security of Logic Locking”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.9 (2016), p. 1411-1424 (cf. p. 9, 28, 32, 40, 42, 43, 47, 77, 90).

[Yas+17a] Muhammad YASIN, Abhrajit SENGUPTA, Mohammed Thari NABEEL et al. “Provably-Secure Logic Locking : From Theory To Practice”. In : *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. New York, NY, USA : Association for Computing Machinery, 2017, 1601–1618 (cf. p. 36).

[Yas+17b] Muhammad YASIN, Abhrajit SENGUPTA, Benjamin Carrion SCHAFER et al. “What to Lock ? Functional and Parametric Locking”. In : *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. New York, NY, USA : Association for Computing Machinery, 2017, 351–356 (cf. p. 34).

[YS17] Muhammad YASIN et Ozgur SINANOGLU. “Evolution of logic locking”. In : *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2017, p. 1-6 (cf. p. 26).

[ZJ13] Sharareh ZAMANZADEH et Ali JAHANIAN. “Automatic netlist scrambling methodology in ASIC design flow to hinder the reverse engineering”. In : *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*. 2013, p. 52-53 (cf. p. 29).

[ZJ16a] Sharareh ZAMANZADEH et Ali JAHANIAN. “ASIC Design Protection against Reverse Engineering during the Fabrication Process using Automatic Netlist Obfuscation Design Flow.” In : *ISeCure* 8.2 (2016) (cf. p. 29).

[ZJ16b] Sharareh ZAMANZADEH et Ali JAHANIAN. “Higher security of ASIC fabrication process against reverse engineering attack using automatic netlist encryption methodology”. In : *Microprocessors and Microsystems* 42 (2016), p. 1-9 (cf. p. 29).

[ZT11] Xuehui ZHANG et Mohammad TEHRANIPOOR. “Case study : Detecting hardware Trojans in third-party digital IP cores”. In : *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. 2011, p. 67-70 (cf. p. 5).

[ZG23] Yadi ZHONG et Ujjwal GUIN. “Complexity Analysis of the SAT Attack on Logic Locking”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.10 (2023), p. 3143-3156 (cf. p. 39).

[ZJK17] Hai ZHOU, Ruifeng JIANG et Shuyu KONG. “CycSAT : SAT-based attack on cyclic logic encryptions”. In : *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, p. 49-56 (cf. p. 36).

[Zuc06] David ZUCKERMAN. “Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number”. In : *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’06. New York, NY, USA : Association for Computing Machinery, 2006, 681–690 (cf. p. 19, 59).

A

Définition des portes logiques avec deux inconnues de la Section 4.2

Cette annexe contient les tableaux de propagation de la logique à deux inconnues déinie dans la Section 4.2.

S	$NOT(S)$
0	1
1	0
X	-X
Y	-Y
-X	X
-Y	Y

S	$BUFFER(S)$
0	0
1	1
X	X
Y	Y
-X	-X
-Y	-Y

Tableau A.1. : Tableaux de propagation de la porte NOT et BUFFER

S_1	S_2	$AND(S_1, S_2)$
0	0	0
0	1	0
0	X	0
0	-X	0
0	Y	0
0	-Y	0
1	0	0
1	1	1
1	X	X
1	-X	-X
1	Y	Y
1	-Y	-Y
X	0	0
X	1	X
X	X	X
X	-X	0
X	Y	Y
X	-Y	-Y
-X	0	0
-X	1	-X
-X	X	0
-X	-X	-X
-X	Y	Y
-X	-Y	-Y
Y	0	0
Y	1	Y
Y	X	Y
Y	-X	Y
Y	Y	Y
Y	-Y	0
-Y	0	0
-Y	1	-Y
-Y	X	-Y
-Y	-X	-Y
-Y	Y	0
-Y	-Y	-Y

S_1	S_2	$NAND(S_1, S_2)$
0	0	1
0	1	1
0	X	1
0	-X	1
0	Y	1
0	-Y	1
1	0	1
1	1	0
1	X	-X
1	-X	X
1	Y	-Y
1	-Y	Y
X	0	1
X	1	-X
X	X	-X
X	-X	1
X	Y	-Y
X	-Y	Y
-X	0	1
-X	1	X
-X	X	1
-X	-X	X
-X	Y	-Y
-X	-Y	Y
Y	0	1
Y	1	-Y
Y	X	-Y
Y	-X	-Y
Y	Y	-Y
Y	-Y	1
-Y	0	1
-Y	1	Y
-Y	X	Y
-Y	-X	Y
-Y	Y	1
-Y	-Y	Y

Tableau A.2. : Tableaux de propagation de la porte AND et $NAND$

S_1	S_2	$OR(S_1, S_2)$
0	0	0
0	1	1
0	X	X
0	-X	-X
0	Y	Y
0	-Y	-Y
1	0	1
1	1	1
1	X	1
1	-X	1
1	Y	1
1	-Y	1
X	0	X
X	1	1
X	X	X
X	-X	1
X	Y	Y
X	-Y	-Y
-X	0	-X
-X	1	1
-X	X	1
-X	-X	-X
-X	Y	Y
-X	-Y	-Y
Y	0	Y
Y	1	1
Y	X	Y
Y	-X	Y
Y	Y	Y
Y	-Y	1
-Y	0	-Y
-Y	1	1
-Y	X	-Y
-Y	-X	-Y
-Y	Y	1
-Y	-Y	-Y

S_1	S_2	$NOR(S_1, S_2)$
0	0	1
0	1	0
0	X	-X
0	-X	X
0	Y	-Y
0	-Y	Y
1	0	0
1	1	0
1	X	0
1	-X	0
1	Y	0
1	-Y	0
X	0	-X
X	1	0
X	X	-X
X	-X	0
X	Y	-Y
X	-Y	Y
-X	0	X
-X	1	0
-X	X	0
-X	-X	X
-X	Y	-Y
-X	-Y	Y
Y	0	-Y
Y	1	0
Y	X	-Y
Y	-X	-Y
Y	Y	-Y
Y	-Y	0
-Y	0	Y
-Y	1	0
-Y	X	Y
-Y	-X	Y
-Y	Y	0
-Y	-Y	Y

Tableau A.3. : Tableaux de propagation de la porte OR et NOR

S_1	S_2	$XOR(S_1, S_2)$
0	0	0
0	1	1
0	X	X
0	-X	-X
0	Y	Y
0	-Y	-Y
1	0	1
1	1	0
1	X	-X
1	-X	X
1	Y	-Y
1	-Y	Y
X	0	X
X	1	-X
X	X	0
X	-X	1
X	Y	0
X	-Y	0
-X	0	-X
-X	1	X
-X	X	1
-X	-X	0
-X	Y	0
-X	-Y	0
Y	0	Y
Y	1	-Y
Y	X	0
Y	-X	0
Y	Y	0
Y	-Y	1
-Y	0	-Y
-Y	1	Y
-Y	X	0
-Y	-X	0
-Y	Y	1
-Y	-Y	0

S_1	S_2	$XNOR(S_1, S_2)$
0	0	1
0	1	0
0	X	-X
0	-X	X
0	Y	-Y
0	-Y	Y
1	0	0
1	1	1
1	X	X
1	-X	-X
1	Y	Y
1	-Y	-Y
X	0	-X
X	1	X
X	X	1
X	-X	0
X	Y	1
X	-Y	1
-X	0	X
-X	1	-X
-X	X	0
-X	-X	1
-X	Y	1
-X	-Y	1
Y	0	-Y
Y	1	Y
Y	X	1
Y	-X	1
Y	Y	1
Y	-Y	0
-Y	0	Y
-Y	1	-Y
-Y	X	1
-Y	-X	1
-Y	Y	0
-Y	-Y	1

Tableau A.4. : Tableaux de propagation de la porte XOR et $XNOR$

B

Preuve des lemmes de la Section 4.3.3

Cette annexe contient les preuves des lemmes de la Section 4.3.3, nécessaire pour prouver que WCC est $NP\text{-}DUR$ et $NON\text{-}APX$.

Lemme 4.3.2. $\forall x \in \mathbb{R}^+, x \geq 2, \forall a \in \mathbb{R}^+, a \geq 2, x + a \leq xa.$

Démonstration. Soit $x \in \mathbb{R}^+$ et $\forall a \in \mathbb{R}^+$,

$$x + a \leq xa \iff a \leq x(a-1) \iff \frac{a}{a-1} \leq x \implies \forall a \geq 2, \frac{a}{a-1} \leq 2 \leq x$$

□

Lemme 4.3.3. $\forall x \in \mathbb{R}^+, x \geq 16$, $log_2(x) \leq \sqrt{x}$

Démonstration. Soit $x \in \mathbb{R}^+$, $\exists k \in \mathbb{R}^+$, $x = 2^k$.

$$log_2(x) \leq \sqrt{x} \iff log_2\left(2^k\right) \leq \sqrt{2^k} \iff k \leq 2^{\frac{k}{2}}$$

Remarquons que $k = 2^{\frac{k}{2}}$ pour $k = 2$ et $k = 4$. Soit $f(k) = 2^{\frac{k}{2}} - k$. Soit $f'(k) = ln(2) \times 2^{\frac{k}{2}-1} - 1$.

$$\begin{aligned}
f'(k) \geq 0 &\iff ln(2) \times 2^{\frac{k}{2}-1} - 1 \geq 0 \\
&\iff ln(2) \times 2^{\frac{k}{2}-1} \geq 1 \\
&\iff 2^{\frac{k}{2}-1} \geq \frac{1}{ln(2)} \\
&\iff \frac{k}{2} - 1 \geq log_2\left(\frac{1}{ln(2)}\right) \\
&\iff k \geq 2 \times \left(log_2\left(\frac{1}{ln(2)}\right) + 1\right) \approx 3.057 \\
&\impliedby k \geq 3.1
\end{aligned}$$

$f'(k)$ est positif pour $k \geq 3.1$, donc $f(k)$ est croissante pour $k \geq 3.1$. Comme $f(k)$ est nulle pour $k = 4$ et la fonction est croissante pour $k \geq 3.1$, $f(k) \geq 0$ pour tous $k \geq 4$. Ainsi, nous avons

$$log_2\left(2^k\right) \leq \sqrt{2^k} \qquad \forall k \in [4, +\infty[\iff log_2(x) \leq \sqrt{x} \qquad \forall x \in [16, +\infty[$$

□

Lemme 4.3.4. *Pour une partition en cliques donnée P et un entier fixé $\Delta \in \mathbb{N}$ tel que $1 \leq \Delta < \omega(P)$,*

$$\underset{k \in \{0,\ldots,\Delta\}}{\operatorname{argmin}} \left\{2^{\Delta+k} \times (\omega(P) - k) \mid \omega(P) > \Delta \geq k \geq 0\right\} = 0$$

Démonstration. Soit P une partition en cliques et $\Delta \in \mathbb{N}$ un entier fixé tel que $1 \leq \Delta < \omega(P)$. Soit $f(k) = 2^{\Delta+k} \times (\omega(P) - k)$. Nous considérons la propriété $\Pi(x) : f(x) \geq f(0)$, où $x \in \{1, \ldots, \Delta\}$. Nous allons prouver cette affirmation par induction.

— Cas de base :

$$\begin{aligned} f(1) - f(0) &= 2^{\Delta+1} \times (\omega(P) - 1) - 2^{\Delta+0} \times (\omega(P) - 0) \\ &= 2^{\Delta+1} \times \omega(P) - 2^{\Delta+1} - 2^{\Delta} \times \omega(P) \\ &= (2^{\Delta+1} - 2^{\Delta}) \times \omega(P) - 2^{\Delta+1} \\ &= 2^{\Delta} \times \omega(P) - 2^{\Delta+1} \\ &\geq 2^{\Delta} \times 2 - 2^{\Delta+1} = 0 \qquad \text{(car } \omega(P) \geq 2\text{)} \end{aligned}$$

Cela signifie que $f(1) \geq f(0)$, donc l'affirmation est vraie pour le cas de base.

— Hypothèse d'induction : Supposons que $\Pi(y)$ soit vraie pour un certain $y \in \{1, \ldots, \Delta - 1\}$.

— Étape d'induction :

$$\begin{aligned} f(y+1) - f(y) &= 2^{\Delta+(y+1)} \times (\omega(P) - (y+1)) - 2^{\Delta+y} \times (\omega(P) - y) \\ &= 2^{\Delta+(y+1)} \times \omega(P) - 2^{\Delta+(y+1)} \times (y+1)) - 2^{\Delta+y} \times \omega(P) + 2^{\Delta+y} \times y \\ &= (2^{\Delta+(y+1)} - 2^{\Delta+y}) \times \omega(P) + y \times (2^{\Delta+y} - 2^{\Delta+(y+1)}) - 2^{\Delta+(y+1)} \\ &= 2^{\Delta+y} \times \omega(P) + y \times -2^{\Delta+y} - 2^{\Delta+(y+1)} \\ &\geq 2^{\Delta+y} \times (y+2) + y \times -2^{\Delta+y} - 2^{\Delta+(y+1)} \qquad \text{(car } \omega(P) \geq y+2\text{)} \\ &\geq 2^{\Delta+y} \times y + 2^{\Delta+y} \times 2 - 2^{\Delta+y} \times y - 2^{\Delta+(y+1)} \\ &\geq 0 \end{aligned}$$

Cela implique que $f(y+1) \geq f(y)$, qui est supérieur à $f(0)$ selon l'hypothèse d'induction. L'affirmation est vraie pour l'étape d'induction.

Nous concluons que $\Pi(x) : f(x) \geq f(0)$ est vraie pour tous les $x \in \{1, \ldots, \Delta\}$. Cela signifie que

$$\underset{k \in \{0,\ldots,\Delta\}}{\operatorname{argmin}} \left\{ 2^{\Delta+k} \times (\omega(P) - k) \mid \omega(P) > \Delta \geq k \geq 0 \right\} = 0$$

□