Consensus blockchain : incitation des utilisateurs d'un réseau à la participation et à la loyauté

Samuel Masseport

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

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche : Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Consensus blockchain : incitation des utilisateurs d'un réseau à la participation et à la loyauté.

Présentée par Samuel Masseport

Le 12 Octobre 2021

Sous la direction de Rodolphe Giroudeau, Benoît Darties et Jorick Lartigau

Devant le jury composé de

Hamamache Kheddouci	Professeur des universités	LIRIS, Lyon	Président
Johanne Cohen	Directrice de recherche	LRI, Paris-Sud	Rapporteur
David Coudert	Directeur de recherche	I3S, Université Côte d'Azur	Rapporteur
Fabrice Theoleyre	Chargé de recherche	ICUBE, Strasbourg	Examinateur
Jorick Lartigau	Docteur	Pikcio SAS	Invité
Rodolphe Giroudeau	Maître de conférences	LIRMM	Directeur
Benoît Darties	Maître de conférences	LIRMM	Co-directeur

UNIVERSITÉ DE MONTPELLIER

Remerciements

Au terme de ces trois années de thèse, mes premiers remerciements s'adressent à mon directeur de thèse, Rodolphe Giroudeau, qui m'a aidé, aiguillé (et surtout supporté) tout au long de ma thèse en plus de m'avoir permis de me développer intellectuellement et humainement. Preuve qu'un supporter de l'OM et un supporter du PSG peuvent s'attendre et travailler ensemble.

J'aimerais remercier mon co-encadrant en entreprise, Jorick Lartigau, qui est à l'initiative de la thèse et qui m'a donné beaucoup d'idées de consensus et de conseils techniques notamment sur les tests.

Merci également à Benoît Darties, mon co-encadrant et collègue de bureau avec qui je me suis entretenue de nombreuses fois sur divers sujets tout au long de ma thèse.

Je remercie les membres de mon jury qui ont accepté de prendre le temps de lire et d'analyser mon manuscrit : Johanne Cohen, David Coudert, Hamamache Kheddouci et Fabrice Theoleyre. Je les remercie également pour leur présence à ma soutenance ainsi que pour m'avoir donné leurs avis sur le manuscrit et la soutenance.

J'ai eu la chance de côtoyer de nombreuses personnes cours de ces trois dernières années. Je tiens à remercier les doctorants du LIRMM et plus particulièrement ceux des équipes MAORE et MAREL avec qui j'ai partagé de nombreux repas et pauses café. Je remercie également les permanents de l'équipe MAORE.

Même si l'entreprise Pikcio a déposé le bilan avant la fin de ma thèse, elle m'a permis de voir autre chose qu'un laboratoire et surtout d'autres personnes avec qui j'ai pu passer de très bons moments. Merci à tous les membres de Pikcio que j'ai pu côtoyer.

Je souhaite également remercier toutes les personnes, telles que mes proches et mes amis, qui ont contribué de près ou de loin à l'accomplissement de mon projet.

Table des matières

Introduction générale

En 1966 l'agence pour les projets de recherche avancée de défense (DARPA pour Defense Advanced Research Projects Agency) du département de la Défense des États-Unis lance un projet de réseau de transfert de paquets de données : ARPANET (de l'anglais Advanced Research Projects Agency Network). Ce projet voit le jour en 1969, il s'inscrit comme étant le premier réseau de transfert utilisant un système de paquets de données et la première alternative aux réseaux basée sur de la communication par circuits électroniques tel que le téléphone. Le système de transmission par paquets de données mis en place sur ARPANET inspirera de nombreux réseaux et deviendra notamment la base du transfert de données sur Internet.

Dans cette thèse, nous définissons une valeur comme une donnée qui ne peut être possédée que par une seule personne à la fois, de l'argent ou un billet de train par exemple. En effet, si l'on prend le cas d'un réseau bancaire, il est important que quand une personne envoie de l'argent à une autre, la première n'ait plus accès à cet argent. Actuellement, la plupart des réseaux d'échanges de valeurs sont des systèmes centralisés dans lesquels un tiers de confiance est chargé de la gouvernance du réseau. Prenons l'exemple d'un système bancaire. Lorsqu'un utilisateur transfère de l'argent à un autre, la banque reçoit sa demande de transaction, vérifie que son solde soit suffisant puis, si tel est le cas, débite le montant du compte de l'utilisateur émetteur et crédite le compte du receveur. La banque joue un rôle d'arbitre, de gouvernance sur le réseau. Le problème étant que ces tiers de confiance ne sont pas parfaits et peuvent prendre des décisions qui ne sont pas dans l'intérêt de ses utilisateurs. Par exemple, 8 novembre 2016, le premier ministre Indien annonce par surprise via Tweeter que les billets de 500 et 1000 Roupies n'avaient plus cours et donc que près de 86% de la monnaie en circulation ne serait plus utilisable car sans valeur légale.

Toutefois, il est difficile de concevoir un réseau d'échanges de valeurs décentralisé (sans tiers de contrôle) résistant aux attaques, aux pannes et aux utilisateurs malveillants. Pour qu'un tel réseau soit viable, il faut que ses utilisateurs puissent se mettre d'accord sur l'état du réseau et sur les transactions à réaliser ou non. Dans un système décentralisé, un algorithme de consensus est un algorithme prédéfini possédé par l'ensemble des participants du réseau et qui leur permet de se mettre d'accord sur l'état du réseau.

Ce n'est qu'en 2008 qu'une personne (ou un groupe de personnes) sous couvert du pseudonyme Satoshi Nakamoto publie un livre blanc [58] dans lequel il détaille le fonctionnement d'une monnaie numérique sans tiers de contrôle. Autrement dit un système

bancaire sans la banque, dans lequel les échanges se font directement d'un utilisateur à l'autre. Cette monnaie, qu'il appelle Bitcoin, est la première valeur pouvant être échangée d'un utilisateur à l'autre sans passer par un quelconque tiers de contrôle. Le réseau proposé par Satoshi Nakamoto est basé sur la technologie de stockage blockchain que nous présentons dans cette thèse. Afin de donner une idée du fonctionnement simplifiée du réseau Bitcoin, en vulgarisant, on peut dire que tous les utilisateurs possèdent l'historique de l'ensemble des transactions réalisées sur le réseau et, pour qu'un utilisateur puisse réaliser une transaction, il l'a transmet à l'ensemble des utilisateurs puis attend qu'elle soit validée par plus de la moitié du réseau. Les utilisateurs possèdent tous un même algorithme prédéfini qui leur permet de se mettre d'accord sur l'historique des transactions ainsi que sur la validité d'une transaction, il s'agit d'un algorithme de consensus. Nous présentons en détail la définition d'un consensus ainsi que le fonctionnement du réseau Bitcoin dans cette thèse.

La publication de Satoshi Nakamoto a ouvert la voix à de nombreux autres réseaux décentralisés d'échange de monnaies numériques (ou crypto-monnaie), chacun proposant son propre algorithme de consensus. Généralement, le consensus des crypto-monnaie prévoit de récompenser les utilisateurs qui ont un « bon » comportement, autrement dit aux utilisateurs qui permettent au réseau de fonctionner correctement. Ainsi, chaque algorithme de consensus, suivant son implémentation incite les utilisateurs du réseau à un comportement plutôt qu'à un autre. Par exemple, le consensus proposé par le réseau Bitcoin propose de rémunérer les utilisateurs qui fournissent de la puissance de calcul au réseau afin de le sécuriser. Plus la part de puissance fournie par un utilisateur est importante par rapport à la puissance globale du réseau, plus sa rémunération est importante. Les utilisateurs de Bitcoin sont donc incités à fournir toujours plus de puissance de calcul au réseau afin d'augmenter leur gain.

Le consensus Bitcoin présente de nombreux avantages notamment au niveau de la fiabilité et de la sécurité qu'il représente, toutefois, il est loin d'être parfait et présente également de nombreux problèmes. Le principal d'entre eux est que le consensus incite ses utilisateurs à toujours plus de calculs, augmentant ainsi l'énergie électrique dépensée par le réseau. Aujourd'hui on estime que si Bitcoin était un pays, il serait le $32^{\text{ème}}$ plus grand consommateur d'électricité, devant la Nouvelle-Zélande et juste derrière les Émirats arabes unis. Un deuxième inconvénient que l'on peut relever est que le consensus ne permet pas plus de 6 ou 7 transactions par seconde, très peu quand on le compare avec Visa et Paypal qui réalisent respectivement 2315 et 193 transactions dans le même temps.

La présente thèse s'attarde sur un autre défaut du consensus proposé par Bitcoin. Actuellement, dans le cas où un utilisateur met sa puissance de calcul depuis plusieurs années à disposition du réseau Bitcoin et qu'un autre utilisateur arrive sur le réseau avec une puissance de calcul nettement supérieure, alors le premier sera directement moins rémunéré que le second malgré le fait qu'il soit présent depuis plus longtemps et qu'il ait contribué à la pérennité du système jusqu'à maintenant. Par la présente thèse, nous proposons deux nouveaux consensus qui proposent d'inciter ses utilisateurs à la participation et la loyauté.

Le premier chapitre sera consacré à définir les concepts et propriétés nécessaires pour la compréhension des résultats et consensus présentés. Dans la première section consacrée

aux systèmes distribués nous rappelons notamment le concept de système pair-à-pair, les problèmes que représente l'échange de valeurs sur un réseau décentralisé et nous présentons le problème de consensus. Les bases de la cryptographie telles que les fonctions de hachage et la cryptographie asymétrique sont présentées dans la deuxième section. La troisième présente la structure de données blockchain, ses avantages et inconvénients, les différents types de blockchains ainsi que les principaux protocoles que les consensus des réseaux blockchains utilisent. Dans la dernière section nous présentons les bases de la théorie de la complexité avec notamment une introduction à la théorie des graphes, le fonctionnement d'une machine de Turing, la définition d'un problème de décision et d'optimisation, la présentation des classes de complexité ainsi que diverses méthodes de preuve d'appartenance à une de ces classes.

Dans le chapitre deux, nous présentons le réseau Bitcoin et son fonctionnement détaillé. Nous voyons comment Bitcoin utilise les concepts de réseau pair-à-pair, de blockchain et de preuve de travail pour proposer un consensus sécurisant un système décentralisé d'échange de valeurs. Nous concluons ce chapitre en présentant les principaux avantages et inconvénients que représente ce consensus.

Le chapitre trois est consacré au premier consensus que nous avons défini. Ce consensus est proposé pour une blockchain privée et propose d'inciter les utilisateurs à échanger sur le réseau. L'idée générale est que plus un utilisateur échange sur le réseau, plus il sera récompensé. Dans ce chapitre, nous présentons les motivations qui nous ont poussées à imaginer un tel consensus avant d'en présenter son fonctionnement. Le chapitre se termine par une analyse critique du comportement que peuvent avoir des utilisateurs d'un réseau utilisant ce consensus et à sa capacité à résister aux attaques.

Le quatrième chapitre est consacré à la présentation d'un algorithme de consensus pensé pour une blockchain publique et incitant ses utilisateurs à la loyauté. Pour ce consensus nous avons imaginé la preuve d'expérience qui permet à tous les acteurs de la blockchain d'estimer rapidement le travail réalisé par un utilisateur au cours des x derniers jours, mois ou années. Avec ce consensus, plus un utilisateur a travaillé par le passé, plus il est récompensé. Nous présentons ce consensus à partir d'une modification du consensus Bitcoin. Le chapitre est conclu par la présentation de divers tests que nous avons réalisés avec ce consensus.

Le cinquième et dernier chapitre est consacré à l'analyse de la complexité d'un jeu de plateau : Ricochet Robots. Le but du jeu est de déplacer des robots sur une grille jusqu'à une case particulière suivant des règles de déplacement imposées par le jeu. Nous montrons notamment que la version d'optimisation du jeu est Poly-APX-hard, nous proposons une autre version de la preuve de Engels et Kamphans [25] qui montre que la version décisionnelle du jeu est PSPACE-complet puis nous finissons en montrant qu'il est possible de simuler une machine de Turing universelle avec un plateau de taille infinie, autrement dit que la version décisionnelle du jeu devient indécidable si le plateau est infini. Nous voulions à l'origine utiliser ce jeu pour la preuve de travail d'un consensus mais le projet à été abandonné suite aux résultats observés.

Certains résultats de cette thèse ont donné lieu à des publications :

- [49] *Proof of Usage: User-centric consensus for data provision and exchange*. Samuel Masseport, Jorick Lartigau, Benoit Darties et Rodolphe Giroudeau pour la conférence BRAINS'19 à Rio de Janeiro (Brésil).
- [50] *Proof of Usage: User-centric consensus for data provision and exchange*. Samuel Masseport, Jorick Lartigau, Benoit Darties et Rodolphe Giroudeau pour le journal Annals of Telecommunications édité par Springer.
- [48] *Proof of Experience: empowering Proof of Work protocol with miner previous work*. Samuel Masseport, Jorick Lartigau, Benoit Darties et Rodolphe Giroudeau pour la conférence BRAINS'20 à Paris (France).

Les articles [49] et [50] décrivent le fonctionnement du consensus présenté dans le troisième chapitre. L'article [48] est une courte présentation du consensus présenté dans le cinquième chapitre.

I

Préliminaires

Dans ce chapitre préliminaire nous abordons les notions et concepts utilisés dans cette thèse. Nous commençons par introduire les notions de base des systèmes distribués, de la cryptographie et de la blockchain. Nous verrons ensuite, au travers d'une étude de cas, comment la crypto-monnaie Bitcoin combine ces notions afin de proposer un système monétaire public décentralisé. Nous introduisons enfin quelques notions en théorie des graphes et en théorie de la complexité.

1.1 Système distribué

1.1.1 Définitions

Un système informatique distribué correspond à une collection de sites (e.g des ordinateurs, des processeurs, ...) travaillant et communiquant pour atteindre un but commun. Un système est dit distribué s'il possède les propriétés suivantes :

- **Plusieurs sites :** le système doit posséder plusieurs (au moins deux) sites indépendants. Ces sites peuvent appartenir à un unique utilisateur ou être contenus dans un seul ordinateur mais ils doivent être indépendants (e.g dans des threads différents).
- **Communication inter-sites :** les sites du système doivent pouvoir communiquer entre eux par le biais de messages dans une limite de temps finie. Actuellement le délai de communication dépend de contraintes physiques.
- **Espace de stockage disjoints :** chaque site du système doit posséder son propre espace de stockage, avec une adresse différente de celle des autres. Les sites ne partagent pas directement de données, s'ils veulent en partager, ils doivent le faire par le biais de messages.

– **But commun :** les sites du système doivent interagir afin d'atteindre un but commun. Considérons deux sites P et Q d'un réseau. Si P calcule $f(x) = x^2$ pour un ensemble de valeurs de x et Q multiplie un ensemble de nombres par π alors on ne peut pas considérer que P et Q constituent un système distribué s'ils ne communiquent pas. Toutefois, s'ils coopèrent pour calculer un ensemble d'aires de cercles alors le réseau formé par P et Q est un exemple de système distribué.

Les exemples suivants sont des cas pour lesquels l'utilisation de systèmes distribués peuvent être utiles :

– **Sites éloignés géographiquement :** dans certaines situations, il est nécessaire que les sites soient éloignés géographiquement. Par exemple, considérons un réseau de banques, chacune devant maintenir à jour le solde de ses utilisateurs. Les banques vont constituer un système distribué afin de communiquer entre elles pour permettre à leurs utilisateurs de réaliser des transactions inter-banques ainsi que des retraits sur des distributeurs répartis géographiquement.

– **Accélérer le calcul :** lorsqu'un seul processeur est utilisé pour un calcul, la limite physique de la machine est rapidement atteinte. À l'instar du parallélisme, les systèmes distribués peuvent permettre d'augmenter la vitesse de calcul. Le principe consiste à diviser un problème en plusieurs sous-problèmes. Les sous-problèmes sont ensuite répartis et résolus indépendamment sur les sites disponibles. De plus, cette méthode est facilement ajustable en ajoutant ou supprimant un site (e.g. en achetant ou vendant un processeur/ordinateur). Le prix d'un tel système est souvent moins élevé que d'investir dans une seule machine dont la puissance de calcul est plus élevée.

– **Partage de ressources :** le terme de ressource peut aussi bien être utilisé pour des ressources physiques que logicielles. Par exemple, un site A peut utiliser une imprimante connectée à un site B. Autre exemple, un site A peut utiliser la puissance de calcul d'autres sites (B et C par exemple) pour augmenter sa puissance de calcul lorsque A en a besoin et que B et C n'utilisent pas la leur. Dernier exemple : les bases de données distribuées, utilisées notamment par des sites de téléchargement, dans lesquelles chaque utilisateur stocke une partie des données (et les données sont enregistrées sur plusieurs sites). Lorsqu'un site veut accéder aux données, il se connecte aux sites qui les possèdent et peut ainsi les consulter ou les télécharger.

– **Tolérance aux pannes :** lorsqu'un système est totalement centralisé, il ne dépend que du bon fonctionnement d'un seul site et la panne de ce site bloque la totalité du système. De plus, dans un tel système, le site central représente un point faible du réseau et peut devenir la cible principale d'utilisateurs malveillants voulant attaquer ou corrompre le système. Avec un système distribué, il est possible de partager les tâches entre plusieurs sites (voir de les faire exécuter par plusieurs sites) afin de garantir que le réseau continue de fonctionner en cas de panne de l'un de ces sites. Il est également possible de partager la mémoire du site principal sur d'autres sites afin d'avoir des sauvegardes en cas de panne ou de nécessité de revenir à un état antérieur (*rollback*) du système.

1.1.2 Système pair-à-pair

Un système pair-à-pair (peer-to-peer en anglais) est un exemple de système distribué dans lequel chaque entité (appelée nœud ou utilisateur) est considérée à la fois comme un client et comme un serveur. La principale différence avec un réseau client-serveur classique porte sur le fait que les nœuds peuvent communiquer directement les uns avec les autres sans passer par un nœud central.

La nature décentralisée des réseaux pair-à-pair augmente la robustesse du réseau car elle supprime le point de défaillance unique qui peut être inhérent à un système basé sur un serveur central. En effet, si un nœud vient à être défaillant (en raison d'une attaque ou d'une panne), le réseau n'est pas compromis. Ainsi, plus il y a de nœuds sur le réseau, plus sa pérennité est garantie. Cette propriété contraste totalement avec un réseau client-serveur pour lequel une panne ou une attaque sur le serveur corrompt totalement le réseau pour les clients.

Un autre avantage important des réseaux pair-à-pair concerne sa scalabilité. Ajouter un nouveau nœud est facile car il n'y a pas besoin de réaliser une configuration sur un serveur central.

Cependant, les réseaux pair-à-pair présentent également certains inconvénients. En effet, l'absence de site central a tendance à supprimer tout tiers de contrôle sur le réseau. Il n'y a donc plus de nœud central capable de modérer le réseau.

De nombreux autres services utilisent des réseaux pair-à-pair :

- réseaux de communications (l'ancienne version de Skype),
- échange de fichiers (BitTorrent, Windows 10 updates, ...),
- crypto-monnaies (Bitcoin, Etereum, ...),
- navigateurs web (Tor),
- etc...

1.1.3 Échange de valeurs

Dans un système d'échange de données, lorsqu'un utilisateur envoie des données à un autre, la donnée est dupliquée. Par exemple, si un utilisateur envoie une photo à un autre par mail, elle est dupliquée et après l'envoi, les deux utilisateurs ont accès à la photo - l'envoyeur à toujours accès à la photo -. Lorsqu'il s'agit d'envoi de valeurs (telles que de l'argent), il est très important que seul l'utilisateur recevant la valeur puisse y avoir accès. Autrement dit, lorsque un utilisateur envoie de l'argent à un autre, il ne faut plus que l'envoyeur ait accès à cet argent. Une transaction est un échange de valeurs entre deux (ou plusieurs) sites d'un réseau.

Second problème, dans un système d'échange de valeurs, les transactions doivent être des opérations atomiques. Autrement dit, lorsqu'une transaction commence à être exécutée, elle doit pouvoir s'exécuter entièrement sans être interrompue avant la fin de son déroulement même en cas de panne du système. En effet, si un utilisateur A envoie 200€ à un utilisateur B, le système doit débiter 200€ à A et créditer 200€ à B en une seule et même opération. Si tel n'est pas le cas et qu'une panne se produit entre les deux opérations, la valeur peut se perdre (A a été débité mais B n'a pas été encore crédité) ou être dupliquée (A n'a pas été débité alors que B a été crédité).

En 1983, Haerder et Reuter [31] définissent les propriétés ACID (atomicité, cohérence, isolation et durabilité) qui garantissent qu'une transaction est exécutée de façon fiable :

- **Atomicité :** garantie qu'une transaction se fait entièrement ou pas du tout. Si le débit ou le crédit d'une transaction ne peut pas être réalisé (manque de fond du compte de débit, panne du système, etc), alors la transaction doit être totalement supprimée, comme si elle n'avait jamais eu lieu.

- **Cohérence :** assure que toutes transactions amènent le réseau d'un état valide à un autre état valide. Une transaction est réalisée si et seulement si le nouvel état du réseau est valide (en accord avec les règles du réseau). Par exemple, si un réseau interdit les soldes négatifs, alors un utilisateur ne pourra pas effectuer de transaction supérieure aux fonds disponibles sur son compte.

- **Isolation :** les transactions doivent s'exécuter indépendamment les unes des autres. Les transactions doivent être exécutées une à une. Considérons un utilisateur A avec un solde de 100€ et un réseau interdisant les soldes négatifs, A ne doit pas pouvoir réaliser deux transactions de débit de 75€ en même temps sinon son solde passera en négatif et la propriété de cohérence ne sera plus respectée. Dans ce cas, le système doit réaliser la première transaction, puis refuser la seconde.

- **Durabilité :** lorsqu'une transaction à été effectuée, elle le reste de façon permanente. Autrement dit, il est impossible de modifier ou supprimer une transaction, si le système veut annuler une transaction, il doit réaliser la transaction inverse.

Double dépense

Dans un système d'échange de valeurs, lorsqu'un utilisateur parvient à dépenser plus d'une fois la même valeur, il y a un problème de double dépense. Cette double dépense peut mener à l'inflation en créant une nouvelle quantité de valeur qui n'existait pas auparavant, elle a tendance à dévaluer la valeur et à faire baisser la confiance des utilisateurs.

Il est facile de se prémunir des problèmes de double dépense lorsqu'il s'agit d'un système centralisé possédant un tiers de confiance qui valide ou invalide chaque transaction une à une. En effet, ce tiers de confiance joue un rôle d'arbitre sur le réseau, il tient à jour l'historique des transactions, le solde de chaque compte et peut décider à lui seul si une transaction est valide ou non.

Lorsque le système est décentralisé, plusieurs sites doivent conserver une copie de l'historique des transactions ainsi que le solde de chaque utilisateur. Les sites doivent alors s'accorder sur les transactions à ajouter à l'historique et recalculer le solde des utilisateurs sachant que les sites ne reçoivent pas forcément toutes les transactions et pas forcément dans le même ordre. De plus, si l'accès à ces sites est public, il se peut que certains d'entre eux soient malveillants et tentent de modifier l'historique ou les soldes à leur avantage. La plupart des systèmes décentralisés résolvent ce problème à l'aide d'un algorithme de consensus (voir sous-section 1.1.4). Les premiers systèmes distribués prévenant la double dépense sont publiés en 2007 par Hoepman [34] et en 2008 par Osipkov [61].

1.1.4 Problème du consensus

Historique : problème des généraux byzantins

Dans certains systèmes distribués, les sites peuvent avoir besoin de se mettre d'accord sur la valeur d'une donnée. Il s'agit d'un problème simple quand tous les sites du réseau sont honnêtes et fonctionnent correctement mais il devient plus difficile si des sites deviennent défaillants ou malveillants. Une faille byzantine intervient dans un système distribué lorsque ses processeurs sont incapables de se mettre d'accord sur la valeur d'une variable commune. Le nom de byzantin vient du problème des généraux byzantins que nous introduisons ci-après.

La résistance aux fautes byzantines (BFT pour Byzantine Fault Tolerance) d'un système distribué à été abordé en profondeur par Lamport et al. [46] en 1982. Dans cette publication, les auteurs introduisent *le problème des généraux byzantins* que l'on peut énoncer de la façon suivante : n généraux byzantins assiègent une ville ennemie et doivent se mettre d'accord sur le plan à suivre pour attaquer la ville mais ils ne peuvent communiquer que par le biais de messagers. Le problème étant qu'il peut y avoir des traîtres parmi eux qui vont essayer de désordonner l'attaque des généraux loyaux. Ils doivent donc avoir un algorithme qui garantit les conditions suivantes :

A - Tous les généraux loyaux décident du même plan.

B - Un petit nombre de traîtres ne doit pas pouvoir décider d'un mauvais plan pour les généraux loyaux.

La condition B est complexe à modéliser car il faut préciser la notion de « mauvais plan ».

Soit v_i le plan d'attaque choisi par le général i. Les généraux vont alors communiquer entre eux jusqu'à ce qu'ils possèdent chacun les choix de tous les autres $v_1, \ldots, v_n$. La condition A est assurée car les généraux vont utiliser le même algorithme pour choisir un plan suivant le choix des n généraux. La condition B est respectée en utilisant une méthode robuste. Par exemple le vote à la majorité. Dans ce cas là si tous les généraux loyaux choisissent un plan d'attaque a, un petit groupe de traîtres proposant un plan b

n'influenceront pas le résultat. Dans le cas où il y a autant de généraux loyaux votant pour a et pour b, un seul traître peut alors choisir le plan à exécuter mais dans ce cas aucun des plans ne peut être considéré comme mauvais.

En formalisant un peu plus les conditions A et B, nous pouvons résoudre le problème des généraux byzantins à l'aide d'un algorithme satisfaisant les conditions suivantes :

1. (de A) Tous les généraux loyaux ont à la fin les mêmes informations $v_1, \ldots, v_n$, ou autrement dit, tous les généraux loyaux utilisent la même valeur de v_i pour tout $i \in \{1, \ldots, n\}$.
2. (de B) Si le général i est loyal alors la valeur qu'il partage avec les autres généraux est la valeur v_i.

Les conditions 1 et 2 sont deux conditions qui s'appliquent sur le plan choisi par le général i. On peut donc se restreindre au problème suivant : comment un seul général peut-il transmettre son plan aux autres en respectant les conditions 1 et 2 ? Le problème des généraux Byzantins peut être défini comme suit :

Problème du commandant et des généraux : Un commandant doit envoyer un ordre à ses $n - 1$ lieutenants tel que :

IC1. Tous les lieutenants loyaux reçoivent le même ordre du commandant.

IC2. Si le commandant est loyal, alors tous les lieutenants loyaux reçoivent l'ordre que le commandant envoie.

IC1 et IC2 sont appelées des conditions de cohérence interactive (interactive consistency conditions en anglais). À noter que si le commandant est loyal alors IC2 implique IC1 (autrement le commandant n'est pas loyal).

Revenons sur le problème de départ qui était de permettre à des sites de prendre une décision dans un système distribué sans qu'un petit nombre de sites malveillants ne puisse perturber ou modifier la décision. Pour répondre à ce problème, il faut que chaque site i partage sa valeur v_i en utilisant une solution au problème des généraux Byzantins dans lequel il joue le commandant donnant l'ordre "utilise v_i pour ma valeurs' aux autres sites qui eux jouent le rôle de lieutenant.

Après avoir montré que le problème des généraux byzantins peut être résolu en résolvant le problème du commandant et des généraux, Lamport et al. [46] considèrent deux modèles, le premier avec messages oraux et le second avec messages signés.

Modèle avec messages oraux :

- Lorsqu'un message est envoyé, il ne peut pas être modifié avant sa réception.
- Un message peut être perdu, mais la perte d'un message peut être détectée.

– Quand un message est reçu (ou que son absence est détectée), le receveur connaît l'envoyeur (ou le site défaillant).

Le modèle avec messages signés satisfait les mêmes conditions que le modèle avec messages oraux plus deux conditions supplémentaires :

– La signature d'un général loyal ne peut pas être modifiée, toute tentative de falsification peut être détectée.

– N'importe qui peut vérifier l'authenticité d'une signature.

Considérant n le nombre de généraux et m le nombre de traîtres parmi eux, Lamport et al. [46] ont montré les deux théorèmes suivant :

Théorème 1 : Le problème des généraux byzantins avec messages oraux admet un algorithme satisfaisant IC1 et IC2 si et seulement si $n > 3m$.

Théorème 2 : Le problème des généraux byzantins avec messages signés admet un algorithme satisfaisant IC1 et IC2 si et seulement si $n \geq m + 2$.

Formalisation

Dans le langage courant, un consensus est un accord des volontés sans aucune opposition formelle. Le consensus se distingue de l'unanimité qui met en évidence la volonté manifeste de tous les membres dans l'accord. Un consensus caractérise l'existence parmi les membres d'un groupe d'un accord général (tacite ou manifeste) et positif pouvant permettre de prendre une décision ou d'agir ensemble sans vote préalable ou délibération particulière. Dans un système centralisé, la gouvernance est généralement assurée par un tiers de confiance connu de tous jouant le rôle d'arbitre, de modérateur sur le système. Par exemple, dans un système bancaire, la banque choisit à elle seule si une transaction est valide ou non et recalcule seule le solde des comptes concernés.

Dans un système décentralisé c'est le consensus qui définit le mode de gouvernance du réseau. Il est défini lors de la création du réseau et permet aux utilisateurs de se mettre d'accord sur son état et ce malgré la présence de pannes ou d'utilisateurs malveillants. Le consensus est distribué à l'ensemble des utilisateurs, il est donc connu de tous.

Plus formellement, le problème de consensus en théorie de calcul distribué est un problème dérivé du problème des généraux byzantins dont l'objectif est d'assurer une certaine fiabilité sur les décisions prises par le système malgré la présence de pannes. Soit un système distribué composé de n sites $\{0, 1, \ldots, n-1\}$ et S un ensemble. Chaque site possède une variable v initialisée tel que $v \in S$. L'objectif du problème est de trouver un algorithme permettant aux sites de se mettre d'accord sur la valeur de v en dépit des dysfonctionnements de certains sites. La valeur finale de v devra respecter les trois conditions suivantes :

- **Terminaison :** tous les sites fonctionnant correctement doivent choisir une valeur finale de v dans une limite de temps finie.

- **Intégrité :** si tous les sites fonctionnant correctement possèdent la même valeur initiale v_i pour v, alors la décision finale doit être v_i.

- **Accord :** les décisions finales de la valeur de v doivent être identiques chez tous les sites fonctionnant correctement.

Modélisation

Considérons un système distribué SD comme un système de transition d'états. Un consensus peut être considéré comme une fonction de transition d'états $ChangeState(E, T) = E'$ prenant en entrée un état consistant de SD et T un ensemble d'opérations à effectuer sur le réseau et retourne un nouvel état consistant E'. Un état consistant de SD est un état dans lequel toutes les règles de SD sont respectées.

Considérons un système bancaire classique qui n'accepte pas les soldes négatifs. E est l'ensemble des soldes des comptes de ses clients et T une liste de transactions à effectuer. Une transaction $t \in T$ correspond à un transfert de x\$ du compte A vers le compte B. La banque vérifie la validité des opérations de T quand elle effectue la fonction $ChangeState$. Ci-après un exemple d'exécution de la fonction $ChangeState$:

$$ChangeState(\{A : 50\$;\ B : 20\$\},\ \{A \text{ envoie } 10\$ \text{ à } B\}) = \{A : 40\$;\ B : 30\$\}$$

Si une opération $t \in T$ n'est pas valide :

$$ChangeState(\{A : 40\$;\ B : 40\$\},\ \{A \text{ envoie } 50\$ \text{ à } B\}) = \text{ERREUR}$$

Le système ne change alors pas d'état. Pour tester la validité des transactions de T la banque utilise une fonction $Verify(T)$ qui renvoie *vrai* si l'ensemble des transactions de T sont valides et *faux* sinon. Dans notre exemple, la banque vérifie simplement que le compte A possède au moins autant d'argent que ce qu'il envoie.

1.2 Cryptographie

La cryptographie est la pratique et l'étude de techniques visant à sécuriser la communication en présence de tierces parties malveillantes. Autrement dit, la cryptographie consiste à construire et à analyser des protocoles qui empêchent des tiers malveillants ou le public de lire des messages privés.

Dans ce large domaine nous présentons seulement les fonctions de hachages et leurs utilisations ainsi que le système de cryptographie asymétrique.

1.2.1 Fonction de hachage

Une fonction de hachage est utilisée pour faire correspondre des données de taille arbitraire à des valeurs de taille fixe. Autrement dit, une fonction de hachage prend en entrée des données (fichier, image, texte, vidéo, etc ...) de taille quelconque, les transforment et retourne de nouvelles données de taille fixe. Les données retournées par une fonction de hachage est appelée le hash, il représente l'empreinte numérique des données en entrée. Les fonctions de hachage sont déterministes et sont généralement utilisées en cryptographie pour signer des fichiers, stoker des mots de passes, vérifier l'intégrité de fichiers ou encore comparer des fichiers.

Étant donné qu'une fonction de hachage prend généralement des données en entrée de taille supérieure à celle en sortie, plusieurs entrées retournent le même hash (principe des tiroirs). Lorsqu'une fonction de hachage retourne le même hash pour deux entrées différentes, on dit qu'il y a une collision entre ces deux entrées. Plus formellement, considérons H une fonction de hachage, x et y deux blocs de données non identiques ($x \neq y$). Il existe une collision entre x et y si et seulement si $H(x) = H(y)$.

La résistance aux collisions d'une fonction de hachage peut être calculée grâce au paradoxe des anniversaires [51]. Considérons une fonction de hachage H qui retourne un hash de n bits pour toutes entrées. H a donc 2^n sorties possibles. On considère que H distribue ses hashs uniformément dans l'espace à sa disposition. En appliquant le paradoxe des anniversaires, si l'on souhaite trouver une collision avec plus de 50% de chance de réussite, on aura besoin de tester au moins $\sqrt{2^n}$ (ou $2^{n/2}$) blocs de données. La résistance aux collisions ne signifie donc pas qu'il n'y en a pas mais qu'elles sont difficiles à trouver.

Une fonction de hachage H est considérée comme solide si :

- étant donné un hash h, il est difficile de trouver une donnée d telle que $H(d) = h$ (i.e. la fonction de hachage est à sens unique),
- étant donné une entrée d_1, il est difficile de construire une entrée d_2 différente de d_1 telle que $H(d_1) = H(d_2)$ (i.e. à partir d'une entrée donnée il est difficile de construire une seconde entrée différente de la première telle que la fonction de hachage retourne le même hash pour les deux entrées),
- il est difficile de construire deux entrées différentes d_1 et d_2 telles que $H(d_1) = H(d_2)$ (i.e. il est impossible de construire deux entrées différentes telles que les deux entrées produisent le même hash).

Cependant il n'existe pas de preuve montrant qu'une fonction de hachage est suffisamment complexe pour être considérée comme solide. On considère donc une fonction de hachage comme solide tant qu'aucun des points précédents n'a été remis en question autrement dit s'il n'existe pas de méthode plus rapide que l'attaque par force brute pour les infirmer et si cette attaque ne peut pas être réalisée en temps raisonnable. Dans ce document, quand nous parlerons de hash, il sera sous entendu qu'il a été calculé avec une fonction de hachage solide.

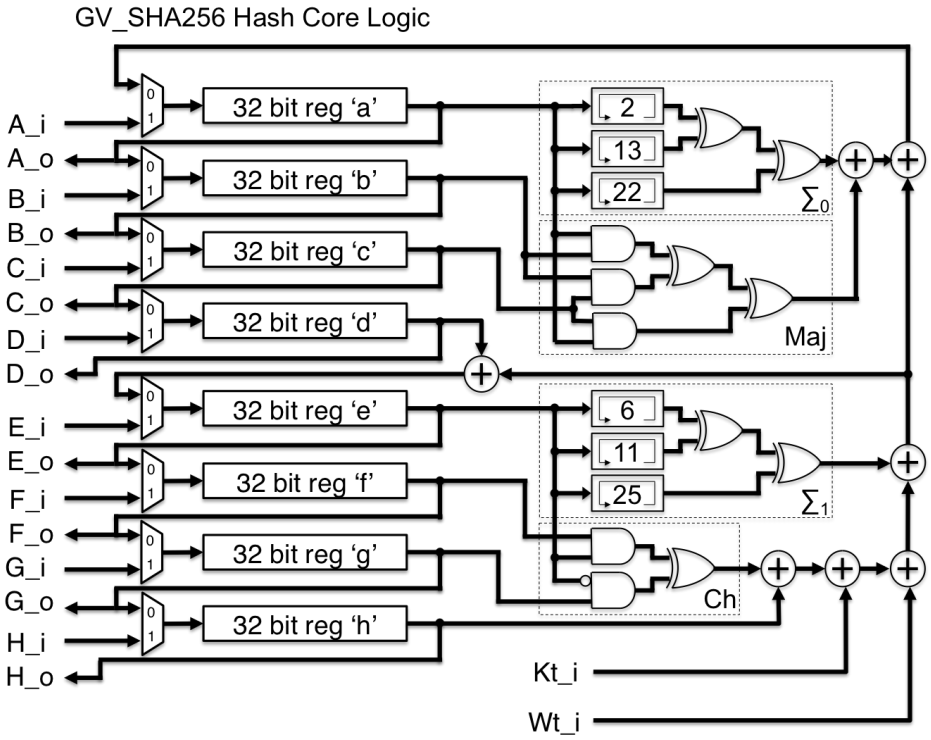


FIGURE 1.1 – *Schéma logique de la fonction de hachage SHA-256.*

SHA-256 [1] est une fonction de hachage trés utilisée en cryptographie. Elle est basée sur une suite de portes logiques (voir figure 1.1) dont on ne détaillera pas le fonctionnement et retourne une suite de 256 bits pour n'importe quelle donnée en entrée (de taille maximale de 2^{64} bits). La résistance à la collision du SHA-256 est estimée à 2^{128} selon le paradoxe des anniversaires, c'est à dire qu'il faudra hacher 2^{128} entrées différentes pour avoir 50% de chances de trouver une collision entre deux de ces entrées. D'autres fonctions de hachages comme MD5 [2] et SHA-1 [3] par exemple sont également beaucoup utilisées.

Salage

Le salage est une méthode permettant de renforcer la sécurité des données destinées à être hachées en leur concaténant des données supplémentaires (appelées sel ou *nonce*) afin d'empêcher que deux informations identiques produisent le même hash. Le salage a pour but de lutter contre les attaques par analyse fréquentielle [4] et les attaques utilisant les tables arc-en-ciel [59]. Le salage peut également permettre de prévenir les attaques par dictionnaire [4] et les attaques par force brute [5] mais seulement s'il n'est pas connu par l'attaquant.

Par exemple, au lieu de hacher directement le mot de passe d'un utilisateur, il est possible de hacher la concaténation du mot de passe avec une autre chaîne de caractères

1. https://fr.wikipedia.org/wiki/SHA-2
2. https://fr.wikipedia.org/wiki/MD5
3. https://fr.wikipedia.org/wiki/SHA-1
4. https://en.wikipedia.org/wiki/Frequency_analysis
5. https://en.wikipedia.org/wiki/Brute-force_attack

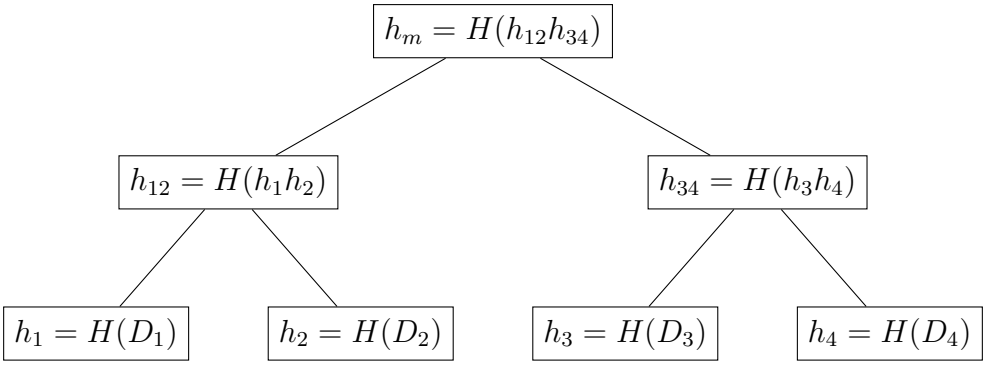


FIGURE 1.2 – *Construction d'un arbre de Merkle binaire.* D_1, D_2, D_3 *et* D_4 *sont les données à inclure dans l'arbre de Merkle et* H *est une fonction de hachage quelconque. Le hash* h_m *est la racine de Merkle obtenue par la construction de l'arbre.*

pseudo-aléatoire. Cette chaîne de caractères peut être obtenue par exemple par un hachage de l'identifiant de l'utilisateur concaténé avec son mot de passe ou bien par une opération binaire entre son identifiant et son mot de passe. Si deux utilisateurs possèdent le même mot de passe, cette opération permet d'éviter que leurs empreintes numériques soient identiques.

Ainsi, si un attaquant tente de retrouver le mot de passe d'un utilisateur, il faut non seulement qu'il inverse la fonction, mais il doit encore explorer l'ensemble des possibilités induites par la présence du sel. Si l'attaque réussit, il doit encore retirer le sel du mot de passe.

En l'absence de salage, il est possible de cracker le système à l'aide de tables de hachage correspondant à des valeurs (telles que des mots de passe) souvent utilisées, par exemple les tables arc-en-ciel [59].

Arbre de Merkle

Un arbre de Merkle [53], aussi appelé arbre de hachage, est une structure de données concaténant des hashs de données. Soit n le nombre de données $D_1, ..., D_n$ à inclure dans l'arbre. Pour construire un arbre de Merkle binaire (voir figure 1.2) on commence par hacher les données une à une à l'aide d'une fonction de hachage H afin d'obtenir n hashs $h_1 = H(D_1), ..., h_n = H(D_n)$. Ensuite, à chaque étape, on calcule le hash de la concaténation de deux hashs de l'étape précédente jusqu'à n'avoir plus qu'un seul hash. Ce dernier hash est appelé la racine de Merkle.

L'intérêt d'un arbre de Merkle est de pouvoir vérifier l'intégrité d'un ensemble de données sans les avoir nécessairement toutes. Par exemple, dans la figure 1.2, nous pouvons vérifier l'intégrité des données D_4 en connaissant seulement h_3, h_{12} et h_m. Considérons D'_4 les données à vérifier, on calcule $h'_4 = H(D'_4)$ puis $h'_{34} = H(h_3h'_4)$ et enfin $h'_m = H(h_{12}h'_{34})$. Si $h'_m = h_m$ alors $h'_{34} = h_{34}$ et $h'_4 = h_4$, donc $D'_4 = D_4$, les données sont vérifiées. Cette vérification marche seulement si l'on fait l'hypothèse qu'il n'y a pas de collision pour la fonction H entre D_4 et D'_4, entre h_3h_4 et $h_3h'_4$ et entre $h_{12}h_{34}$ et $h_{12}h'_{34}$ mais les collisions sont si rares dans les fonctions de hachage (par exemple pour la fonction SHA-256, une

chance sur 2^{256}) que l'on peut considérer cette hypothèse comme vraie.

1.2.2 Cryptographie asymétrique

La cryptographie asymétrique (ou cryptographie à clef publique / clé privée) est un système cryptographique permettant à deux utilisateurs de communiquer sans qu'un utilisateur tiers qui intercepte le message ne puisse le lire. La cryptographie asymétrique a été imaginée par Whitfield Diffie et Martin Hellman en 1976 [20, 21] sans donner d'exemple de clef publique/privée. Il faut attendre 1978 pour que Ronald Rivest, Adi Shamir et Leonard Adleman en trouve un [63], le RSA, abréviation tirée des trois noms de ses auteurs. Depuis, de nouveaux protocoles basés sur la cryptographie asymétrique ont été définis : RSA, ElGamal [24], Merkle-Hellman [52], courbes elliptiques [56], etc.

Rappelons que, en cryptographie, le chiffrement est un procédé permettant de rendre la compréhension de données impossible à toute personne qui n'a pas la clef de déchiffrement. Le déchiffrement est le procédé inverse, il permet à un utilisateur (qui possède la clef de déchiffrement) de lire des données chiffrées.

En cryptographie asymétrique, chaque utilisateur possède une paire de clefs :

- une clef publique, qui peut être connue par l'ensemble des utilisateurs sans compromettre la sécurité,
- et une clef privée, qui ne doit être connue par personne d'autre que son propriétaire.

Par convention, La clef publique est appelée clef de chiffrement et la clef privée est appelée clef de déchiffrement. Les clefs sont générées par le biais d'algorithmes cryptographiques basés sur des fonctions mathématiques asymétriques (i.e. fonctions dont l'image d'un antécédent donné est facilement calculable mais tels qu'il est difficile de trouver un antécédent à une image donnée). Dans un système de cryptographie asymétrique, la clef publique correspond au résultat de la fonction asymétrique et la clef privée à l'antécédent utilisé pour le calcul.

Dans un système de cryptographie asymétrique, toute personne peut chiffrer un message en utilisant la clef publique du destinataire, mais ce message ne pourra être déchiffré qu'avec la clef privée du destinataire et donc seulement par le destinataire, sous l'hypothèse qu'il soit le seul à connaître sa clef privée. Voir figure 1.3.

La cryptographie asymétrique permet également de vérifier l'authenticité de l'expéditeur. L'expéditeur utilise sa clef privée pour chiffrer la signature numérique d'un message (i.e. son hash) que le destinataire pourra déchiffrer avec la clef publique de l'expéditeur, et uniquement cette dernière. Tout utilisateur possédant la clef publique de l'expéditeur peut déchiffrer le message et le comparer avec une signature numérique revendiquée par l'expéditeur. Si le hash déchiffré correspond au hash du message, alors le message a bien été envoyé par le propriétaire de la clef privée correspondante, l'origine du message est

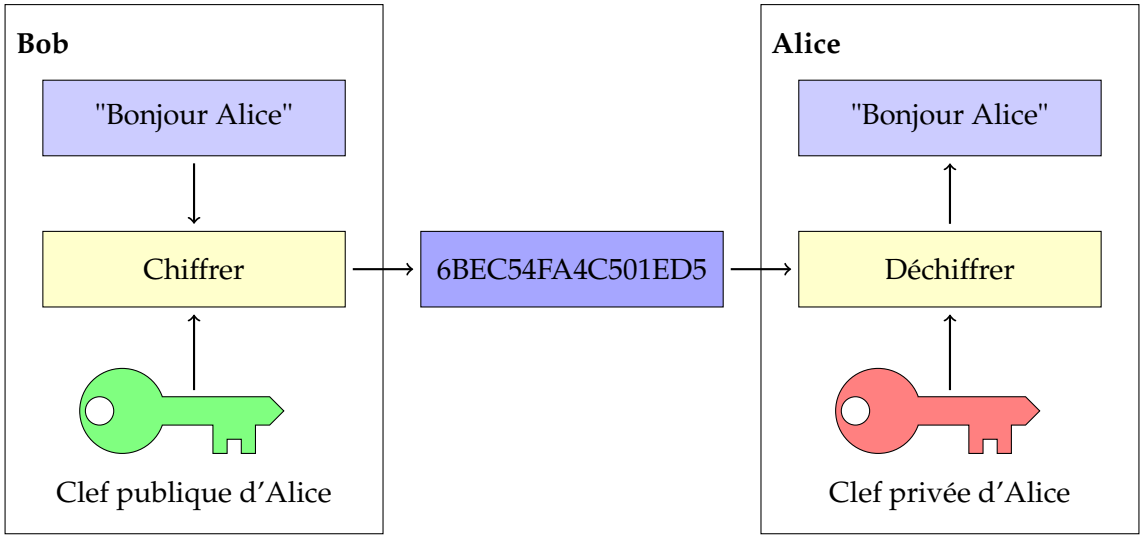


FIGURE 1.3 – *Protocole d'échange de message en cryptographie asymétrique. Bob chiffre son message avec la clef publique d'Alice et envoie le message chiffré. Alice reçoit le message de Bob et le déchiffre grâce à sa clé privée. Un utilisateur malveillant interceptant le message sera incapable de le lire (sous l'hypothèse qu'il ne possède pas la clef privée d'Alice).*

vérifiée (sous l'hypothèse que l'expéditeur soit le seul à connaître sa clef privée) et le message n'a pas été modifié (Voir figure 1.4).

1.3 Blockchain

1.3.1 Définition

À l'origine, la blockchain est une structure de données inventée par Haber et Stornetta [30] en 1990 dans le but de garantir qu'un fichier daté n'a pas été modifié ou antidaté. Plus tard, Bayer, Haber et Stornetta [2] incorporent le concept d'arbre de Merkle au système, permettant ainsi à plusieurs documents d'être enregistrés dans un seul bloc. Depuis 2008 et l'arrivée du Bitcoin [58] la technologie blockchain est principalement utilisée par des réseaux publics proposant un échange de valeurs, souvent de la crypto-monnaie et dans laquelle sont stockées les transactions.

La blockchain [71] (ou chaîne de blocs) est une structure de données qui n'autorise que l'ajout de données (contrairement à une base de données classique qui autorise généralement la modification ainsi que la suppression de données). Les blocs sont ordonnés pour créer une chaîne, ainsi, chaque bloc possède un unique successeur (sauf le dernier bloc de la chaîne) et un unique prédécesseur (sauf le premier bloc de la chaîne). Chaque bloc correspond à un ensemble de données et contient le hash du bloc précédent (voir figure 1.5), à l'exception du premier bloc appelé bloc de genèse. Le fait qu'un bloc contienne le hash du bloc précédent implique "grossièrement" que chaque bloc garantit l'intégrité des blocs précédents de la chaîne : Si un bloc est inséré, modifié ou supprimé en milieu de chaîne, le hash à insérer dans le bloc suivant ne correspond plus, rendant invalide par cascade l'ensemble des blocs suivants (voir sous-section 1.3.2).

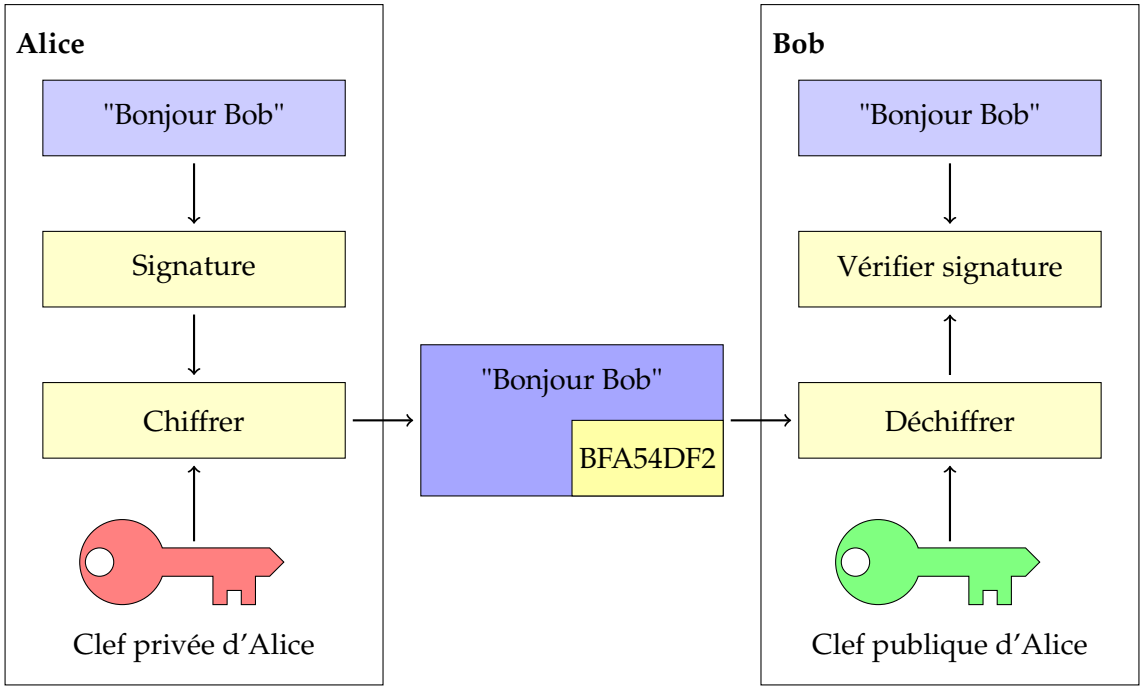


FIGURE 1.4 – *Dans cet exemple seule la signature du message est chiffrée. Alice calcule la signature de son message (grâce à une fonction de hachage par exemple), chiffre la signature avec sa clef privée puis envoie le message d'origine ainsi que la signature chiffrée. Bob déchiffre la signature du message grâce à la clef publique d'Alice et calcule la signature du message avec la fonction utilisée par Alice. Si les deux signatures correspondent, Bob a vérifié que Alice est bien l'expéditrice du message et qu'il n'a pas été modifié.*

1.3.2 Pérennité des blocs

Le fait que chaque bloc contienne le hash du bloc précédent nous permet de garantir que les données sont immuables. En effet si l'on modifie les informations d'un bloc, son hash sera également modifié. Ainsi, puisque le bloc suivant contient ce nouveau hash, le hash du bloc suivant ne correspond plus aux informations qu'il contient, il est considéré comme invalide. Il faudra donc recalculer le hash du bloc ainsi modifié mais cette opération rend le bloc d'après invalide. Il faut de nouveau calculer le hash du bloc d'après et ainsi de suite jusqu'à la fin de la blockchain. Ainsi, plus un bloc est ancien dans la chaîne plus il faudra modifier de blocs pour valider cette modification. Par conséquent, plus un bloc est ancien, plus il est pérenne.

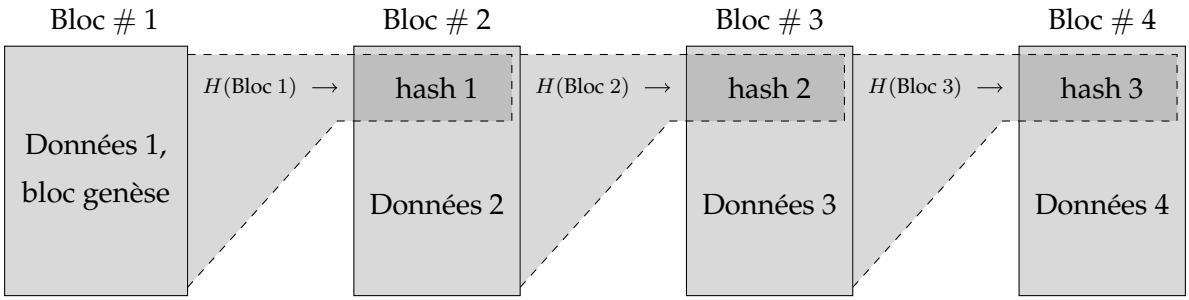


FIGURE 1.5 – *Structure d'une blockchain, chaque bloc contient le hash du bloc précédent. H est une fonction de hachage quelconque.*

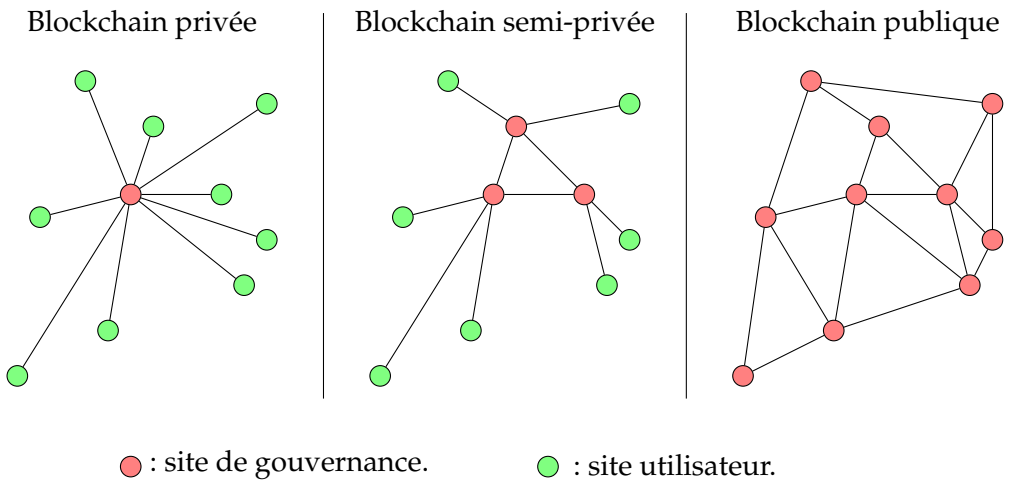


FIGURE 1.6 – *Exemple des trois types de blockchains. De gauche à droite : blockchain privée, blockchain semi-privée, blockchain publique. Les sites de gouvernance (en rouge) sont les nœuds des acteurs qui contrôle et maintiennent le réseau à jour. Les sites utilisateurs (en vert) sont les nœuds utilisés par des utilisateurs « classiques » qui n'ont pas de contrôle sur le réseau.*

1.3.3 Les différents types de blockchain

Blockchain publique : les blockchains publiques sont utilisées sur des réseaux pair-à-pair dont l'accès est public et dans lesquels tous les utilisateurs ont les mêmes pouvoirs. Ils peuvent tous réaliser des transactions, participer sur le réseau et accéder à la blockchain. Les blockchains publiques sont généralement sécurisées en incitant leurs utilisateurs à participer à la vérification des transactions par des avantages économiques. Les blockchains publiques sont gouvernées par des consensus. Les exemples de blockchains publiques les plus connues sont Bitcoin et Ethereum.

Blockchain privée : Dans ces types de blockchains, l'accès et les détails des transactions sont limités à un certain nombre d'utilisateurs. Le processus d'approbation est réservé à un unique acteur, bien que la consultation du registre peut être publique.

Blockchain semi-privée : les blockchains semi-privées (également appelées blockchains sous contrôle ou de consortium) sont gouvernées par un groupe restreint d'acteurs du réseau. Certains utilisateurs peuvent être connus et d'autres anonymes. Son accès peut être privé ou public et le droit d'accès au registre peut être privé, public ou hybride. Les principaux utilisateurs de blockchains semi-privées sont des banques ou des assurances car elles leurs permettent de contrôler la validation des transactions et de connaître l'identité des utilisateurs tout en garantissant la protection de leurs données personnelles.

Les principales différences entre ces trois types de blockchains résident dans la répartition des sites sur le réseau et la répartition du pouvoir de gouvernance. La figure 1.6 présente un exemple de chaque type de blockchain.

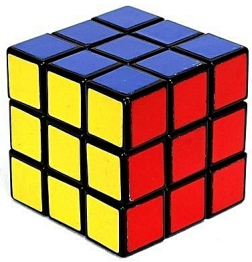


FIGURE 1.7 – *Le Rubik's cube, exemple de preuve de travail de la vie courante.*

1.3.4 Preuve de travail

La preuve de travail [23, 41] (Proof of Work ou PoW en anglais) est un protocole visant à prémunir un service des attaques par déni de service (DoS/DDos) ou d'autres abus comme le spam, en exigeant un travail de la part du demandeur de service. Le travail demandé est généralement un calcul complexe à réaliser, coûteux en temps et en énergie dont le but est de résoudre un problème mathématique asymétrique (soit un problème long à résoudre mais dont la solution est facilement vérifiable par un tiers). Un bon exemple de problème asymétrique pouvant être utilisé comme preuve de travail dans la vie courante est le Rubik's cube (voir figure 1.7). En effet, le Rubik's cube est long à résoudre mais n'importe qui peut vérifier très rapidement s'il est correctement résolu.

Un autre bon exemple de preuve de travail serait le puzzle Eternity II[6], un edge-matching puzzle [18] de 256 pièces carrées aux bords colorés qu'il faut disposer sur une grille de taille 16×16 de sorte que les pièces qui se touchent doivent se toucher par un bord de la même couleur (voir figure 1.8). Ce problème extrêmement compliqué (aucune solution depuis 14 ans malgré la récompense de 2 millions de dollars offerte à la première personne trouvant la solution) peut être vérifié presque instantanément (en vérifiant que les pièces adjacentes ont bien la même couleur de bordure).

La preuve de travail est à la base de la plupart des consensus utilisés par les crypto-monnaies (Bitcoin, Ethereum, ...). Généralement, lorsqu'il s'agit d'un système distribué (tel que celui utilisé pour les crypto-monnaie publiques) les utilisateurs du réseau sont mis en compétition pour calculer une preuve de travail. Le premier à présenter une preuve valide aux autres est élu pour le prochain tour et peut, pour une blockchain, choisir le prochain bloc à ajouter à la chaîne. Le principal problème de ces consensus est qu'ils sont très demandeurs en énergie étant donné que tous les utilisateurs réalisent le calcul d'une preuve de travail et que seul le plus rapide d'entre eux sera choisi.

1.3.5 Preuve d'enjeu

La preuve d'enjeu ou preuve de participation (Proof-of-Stake ou PoS en anglais) est la base de certains consensus de crypto-monnaie. La preuve d'enjeu est introduite en 2012

6. https://en.wikipedia.org/wiki/Eternity_II_puzzle

FIGURE 1.8 – *Le puzzle Eternity II, à gauche le plateau de jeu sur lequel on voit une solution partielle et à droite des pièces du jeu.*

par King et Nadal [44] afin de proposer une solution moins énergivore que la preuve de travail.

Dans les réseaux de crypto-monnaies basés sur la preuve d'enjeu, le créateur du bloc suivant est choisi par une fonction aléatoire pondérée par divers critères telles que la richesse des utilisateurs ou leur ancienneté par exemple. Généralement, la preuve d'enjeu demande à l'utilisateur de prouver la possession d'une certaine quantité de crypto-monnaie afin de prouver sa "participation" au système et ainsi prétendre pouvoir ajouter le bloc suivant. L'utilisateur choisi pourra donc enregistrer le prochain bloc et en sera récompensé.

Il existe un grand nombre de variations de preuve d'enjeu et plus spécifiquement sur la pondération de la fonction de sélection aléatoire. Cette sélection peut être pondérée ou non par un ou plusieurs critères tels que la richesse, l'âge de la monnaie, l'ancienneté de l'utilisateur sur le réseau, etc... Dans certains consensus, il peut il y avoir des conditions (sur la quantité de monnaie ou l'utilisateur) pour autoriser ou non la participation de l'utilisateur. Par exemple, on peut imaginer une preuve d'enjeu pour laquelle seuls les utilisateurs avec au moins un an d'ancienneté et plus de 1000 pièces sur le réseau peuvent être candidats et enregistrer le bloc suivant.

De la pondération de la fonction aléatoire dépend l'incitation du réseau. En effet, selon les critères de pondération et les conditions exigées pour participer au tirage au sort de la

fonction aléatoire, l'incitation ne sera pas la même. Prenons l'exemple d'une preuve d'enjeu pour laquelle tous les utilisateurs participent et dont le tirage est pondéré suivant leur richesse. C'est à dire que la probabilité qu'un utilisateur soit choisi (et donc récompensé) est égale à la proportion de monnaie qu'il a sur le réseau. Un utilisateur possédant 20% de la monnaie globale du réseau aura donc 20% de chance d'être choisi pour ajouter le bloc suivant et aura 10 fois plus de chance d'être choisi qu'un utilisateur ayant seulement 2% de la monnaie globale. Dans ce cas là, la preuve d'enjeu incite ses utilisateurs à économiser leur monnaie car en la dépensant, ils diminuent leurs chances d'être sélectionnés et donc leurs revenus.

1.3.6 Division de la chaîne : le fork

Lorsqu'une blockchain se divise (e.g. un bloc possède plus d'un successeur) et que les sites ne travaillent plus sur le même bloc, on dit que la chaîne a subi un fork (voir figure 1.9). Le problème d'un fork est qu'il engendre une différence d'information sur la chaîne pouvant être critique selon les cas d'usage (e.g. transaction bancaire). On distingue deux catégories de forks :

- **Fork involontaire :** lorsque les données d'un système distribué sont enregistrées sous forme de blockchain et en l'absence d'élection de leader déterministe (i.e. lorsqu'aucune fonction déterministe ne permet au site de choisir celui qui écrit le bloc suivant) il peut arriver que plusieurs sites écrivent sur la chaîne dans le même laps de temps puis partagent respectivement la nouvelle chaîne obtenue à leurs pairs. Dans ce cas là une partie du réseau va considérer un bloc alors que l'autre partie considérera l'autre bloc. Il est important que le consensus prémunisse à terme la blockchain de ce problème afin d'éviter la redondance ou la différence d'information entre les sites. Généralement, lorsque le consensus engendre l'apparition de forks involontaires, le consensus prévoit une règle supplémentaire afin de choisir laquelle des chaînes est la bonne (chaîne la plus longue, la plus utilisée, etc...) afin que les sites, à terme, n'utilisent qu'une seule et même chaîne. Les blocs créés ne faisant plus partie de la chaîne, ils sont abandonnées, on les appelle des blocs orphelins.
- **Fork volontaire :** un fork volontaire apparaît quand une partie des sites choisissent de modifier le consensus de telle sorte qu'à la suite de blocs acceptés par le consensus modifié, on ne puisse plus ajouter de blocs utilisant le consensus de départ (et vice versa). Dans ce cas là un fork est créé avec d'un coté des blocs respectant le consensus de départ et de l'autre des blocs respectant le consensus modifié.

1.4 Problèmes Algorithmiques

En informatique théorique, un problème est un objet mathématique qui représente une question ou un ensemble de questions auxquelles un ordinateur devrait être en mesure de

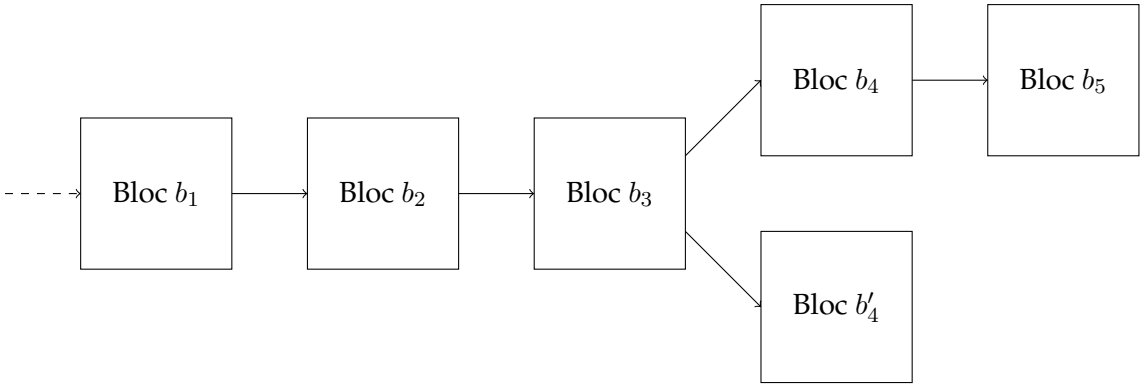


FIGURE 1.9 – *Exemple de fork. Considérons qu'il s'agisse d'un fork involontaire et qu'une règle du consensus est de travailler toujours avec la chaîne la plus longue. Deux sites ont donc enregistrés dans le même laps de temps les blocs b_4 et b'_4. Il y a donc une partie du réseau qui travaille en considérant le bloc b_4 et une autre partie le bloc b'_4. Lorsque le bloc b_5 est enregistré et partagé, la chaîne b_1, b_2, b_3, b_4, b_5 devient plus longue que la chaîne b_1, b_2, b_3, b'_4 et donc l'ensemble des sites vont la considérer. Ainsi, le bloc b'_4 est abandonné, il devient un bloc orphelin.*

répondre. Le plus souvent, ces problèmes sont de la forme : étant donné un objet (l'instance), effectuer une certaine action ou répondre à telle question. On distingue principalement deux types de problèmes : les problèmes de décision et les problèmes d'optimisation que nous détaillons dans la sous-section 1.4.1 et la sous-section 1.4.3 respectivement.

1.4.1 Problème de décision

En informatique théorique, un problème de décision consiste à répondre à une question mathématique soit par "oui" soit par "non". Cette question concerne un objet appelé donnée (ou entrée) du problème. Les données d'un problème peuvent généralement prendre une infinité de valeurs, une instance d'un problème correspond aux données dont on a fixé les valeurs. Par exemple, considérons le problème de décision suivant :

NOMBRE PREMIER

Donnée : Soit n un nombre premier naturel.
Question : Est-ce que n est un nombre premier ?

Dans ce problème, n correspond à la donnée d'entrée du problème, il peut prendre une infinité de valeurs. Considérant que n est fixe, par exemple $n = 12$, alors $n = 12$ est une instance de mon problème et dans ce cas la solution sera "non". Un certificat est un objet permettant de prouver la solution, dans notre cas, le certificat montrant que $12 = 3 \times 4$ suffit pour montrer que notre solution est correcte, que 12 n'est pas premier.

Décidabilité et machine de Turing

Un problème de décision est décidable si et seulement s'il existe un algorithme capable de le résoudre, sinon le problème est indécidable. La notion d'indécidabilité ne s'arrête pas à un simple manque de connaissance (« on ne connaît pas d'algorithmes pouvant résoudre ce problème ») mais à une non-existence d'algorithme permettant de résoudre un problème. Le problème de l'arrêt peut être énoncer comme suit : « étant donné un programme et une entrée finie, est ce que l'exécution du programme se finira (ou est ce qu'il s'exécutera indéfiniment) ? ». Le problème de l'arrêt a été montré indécidable par Turing [69] en 1936 grâce notamment à la définition d'un objet mathématique appelé machine de Turing encore utilisé de nos jours pour montrer qu'un problème est décidable ou non.

Une machine de Turing se compose des éléments suivants :

- **Un ruban infini.** Le ruban correspond à une infinité de cellules (ou case), souvent représenté sur une ligne horizontale, qui contiennent chacune un symbole d'un alphabet donné. L'alphabet est de taille fini et contient un symbole spécial appelé "symbole blanc" ainsi qu'au minimum un autre symbole. Le ruban doit être infini des deux cotés, à gauche comme à droite. Par défaut, on considère que les cases contiennent le symbole blanc.
- **Une tête de lecture/écriture.** La tête de lecture/écriture est toujours positionnée sur une des cellules du ruban. Elle permet de lire le symbole contenu par la cellule ainsi qu'à remplacer le symbole que la cellule contient par un autre symbole de l'alphabet. La tête peut être déplacée vers la gauche ou vers la droite sur le ruban. La cellule sur laquelle se trouve la tête est appelé cellule courante.
- **Un registre d'état.** Le registre d'état mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini, et il existe un état spécial appelé "état de départ" qui est l'état initial de la machine.
- **Une fonction de transition.** La fonction de transition indique à la machine le symbole à écrire sur la cellule où se trouve la tête, comment déplacer la tête et quel état est le nouvel état en fonction du symbole lu par la tête sur le ruban et l'état courant. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête. À noter que la fonction de transition d'état n'est pas forcément déterministe. Autrement dit, pour deux exécutions de la fonction avec le même état courant et le même symbole lu, la machine peut réaliser deux actions différentes.

La machine de Turing possède de nombreuses définitions, nous avons choisi de présenter celle de Hopcroft et al. [36] qui décrivent une machine de Turing M par un 7-uplet $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ avec :

- Q ensemble fini, non-vide d'états,
- Σ ensemble fini de symboles d'entrée,

- Γ ensemble des symboles possibles d'écrire sur le ruban, Σ est toujours un sous ensemble de Γ,
- $\delta(q, X) \rightarrow (p, Y, D)$ est une fonction de transition qui, étant donné un état $q \in Q$ et un symbole X retourne un triplet composé de : l'état suivant p, le symbole Y à écrire dans la cellule sur laquelle la tête se trouve et une direction D, de valeur L ("left", gauche) ou R ("right", droite), dans laquelle la tête se déplace.
- $q_0 \in Q$ l'état initial de M,
- $B \in \Gamma \setminus \Sigma$ symbole blanc, et
- $F \subseteq Q$ ensemble d'états finaux.

Dans une machine de Turing, à chaque étape la machine est dans un état courant q. La tête commence par lire le symbole X qui est écrit sur dans la cellule courante. Après avoir calculé le triplet (p, Y, D) obtenu par la fonction δ avec (q, X) en entrée, l'état courant devient p, la tête écrit Y (à la place de X) dans la case courante puis elle est déplacée sur la cellule de gauche (si $D = L$) ou celle de droite (si $D = R$).

Une machine de Turing est dite universelle si elle peut simuler toute autre machine de Turing peu importe l'entrée. On dit qu'une machine en simule une autre lorsque une machine de Turing universelle M recevant en entrée un codage d'une machine M' et des données Σ, elle produit le même résultat que la machine M' à laquelle on donnerait en entrée les données Σ.

Un langage informatique est dit Turing-complet s'il permet de calculer exactement les problèmes que les machines de Turing peuvent calculer, il est capable de simuler toute machine de Turing. Autrement dit, un langage Turing-complet peut calculer tout ce qui est calculable.

Une machine de Turing universelle est Turing-complète, elle a la capacité de calculer tout ce qui est calculable. En lui fournissant le codage adéquat, elle peut simuler toute fonction récursive, analyser tout langage récursif, et accepter tout langage partiellement décidable. Selon la thèse de Church-Turing, les problèmes résolubles par une machine de Turing universelle sont exactement les problèmes résolubles par un algorithme ou par une méthode concrète de calcul.

Théorie de la complexité

La théorie de la complexité propose d'étudier formellement la quantité de ressources (temps, espace mémoire, etc.) dont a besoin un algorithme pour résoudre un problème algorithmique. Il s'agit donc d'étudier la difficulté intrinsèque des problèmes afin de les organiser par classes de complexité et d'étudier les relations entre celles-ci. Lorsque nous parlons de complexité, nous parlons du temps d'exécution d'un algorithme (ou de l'espace mémoire qu'il utilise) mesuré en terme d'accroissement de la taille des données en entrée.

Les classes de complexité permettent de classifier les problèmes selon certains critères, les plus communs sont le temps (i.e. nombre d'opérations atomiques) et l'espace mémoire.

Les problèmes sont classés suivant la quantité de temps (ou d'espace mémoire) minimale nécessaire pour résoudre un problème dans le pire des cas (i.e. pour la pire des instances, celle qui demande le plus de temps ou d'espace mémoire pour être résolu) en fonction de la taille de l'entrée. Les classes usuelles sont définies en utilisant les machines de Turing comme modèles de calcul en fonction du temps et de l'espace minimal nécessaire pour exécuter la machine de Turing qui résout le problème.

Il existe une multitude de classes différentes. Les classes que nous abordons dans la présente thèse sont les suivantes :

- **P** : pour polynomial. Un problème de décision appartient à la classe **P** s'il est décidable par une machine de Turing déterministe en temps polynomial par rapport à la taille de l'entrée. Autrement dit, **P** est la classe de complexité regroupant l'ensemble des problèmes de décision pouvant être résolus par un algorithme qui s'exécute en temps polynomial en fonction de l'entrée. C'est à dire que si l'on considère une entrée n d'un problème de décision Π, alors, dans le pire des cas, l'algorithme sera exécuté en au plus n^k étapes, avec k une constante.

- **NP** : pour non déterministe polynomial (de nondeterministic polynomial time en anglais). Un problème de décision appartient à la classe **NP** s'il est décidable par une machine de Turing non déterministe en temps polynomial par rapport à la taille de l'entrée. Plus intuitivement, un problème est dans la classe **NP** si l'on peut vérifier un certificat en temps polynomial. Par exemple, pour le problème SATISFAISABILITÉ (SAT, présenté ci-après), il est possible, à partir d'un certificat (une assignation des variables propositionnelles), de vérifier que la formule Φ est satisfaite en temps polynomial.

- **PSPACE** : pour espace polynomial (de polynomial space en anglais). Un problème de décision appartient à la classe **PSPACE** s'il est décidable par une machine de Turing déterministe avec un espace polynomial.

- **NPSPACE** : pour non déterministe espace polynomial (de nondeterministic polynomial space en anglais). Un problème de décision appartient à la classe **NPSPACE** s'il est décidé par une machine de Turing non déterministe avec un espace polynomial. Cette classe est en pratique très peu utilisée car il à été montré par Savitch [67] que la classe **NPSPACE** est équivalante à la classe **PSPACE**, autrement dit **NPSPACE** = **PSPACE**.

SATISFAISABILITÉ (SAT)

Donnée : Une formule de logique propositionnelle Φ.
Question : Existe-t-il une assignation des variables propositionnelles de Φ qui rend la formule vraie ?

Le table 1.1 résume les classes de complexité présenté en fonction de la machine de Turing correspondante.

Critère	Déterministe	Non déterministe
Temps	**P**	**NP**
Espace	**PSPACE**	**NPSPACE**

TABLE 1.1 – *Résumé des caractéristique des classes de complexité.*

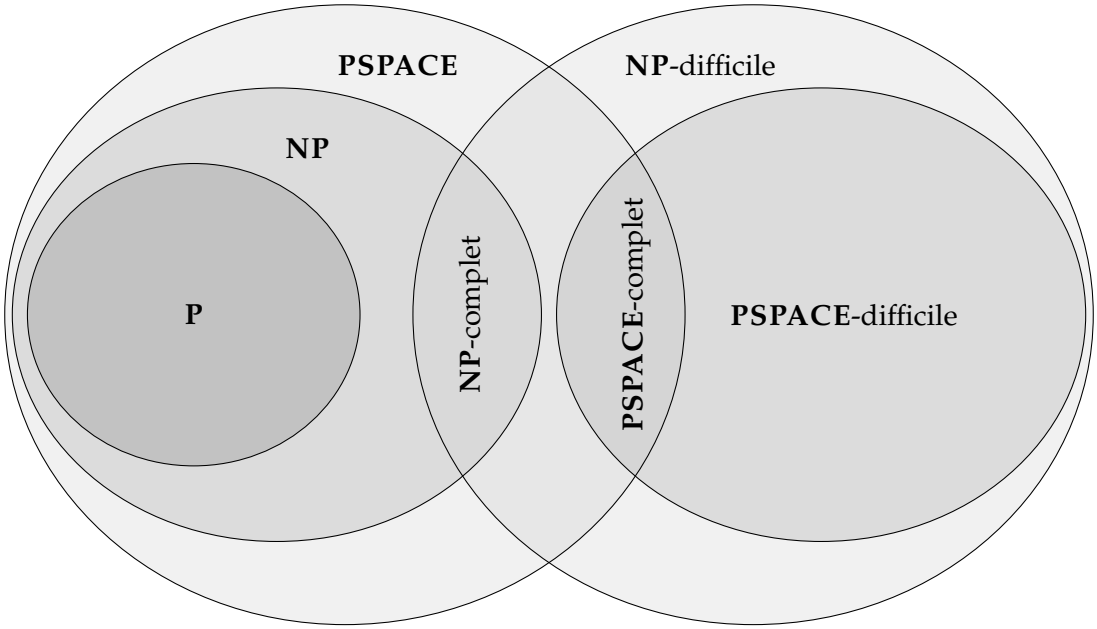


FIGURE 1.10 – *Diagramme de Venn représentant les relations entre les classes de complexité (sous l'hypothèse que* **P** $\neq$**NP**.

1.4.2 Classes C-difficiles et C-complètes

Soit **C** une classe de complexité (telle que *NP* ou *PSPACE* par exemple). On dit qu'un problème est **C**-difficile si ce problème est au moins aussi difficile que tous les problèmes dans **C**. Quand un problème est **C**-difficile et qu'il appartient à la classe **C**, on dit qu'il est **C**-complet. Les problèmes **C**-difficiles sont les majorants de **C**, ils sont au moins aussi difficiles qu'un problème de la classe **C** (et peuvent potentiellement l'être encore plus). Les problèmes **C**-complets sont les plus grands éléments (les plus difficiles) de **C**. La figure 1.10 présente les relations entre les différentes classes présentées précédemment.

Pour montrer qu'un problème appartient à une classe **C**-difficile, on utilise une preuve que l'on appelle réduction. Soit deux problèmes Π et Π'. Une réduction de Π à Π' consiste à trouver un algorithme d'une complexité donnée (inférieure à celle de la classe **C**) transformant toute instance du problème Π' en instance du problème Π. Ainsi, considérant que l'on sache résoudre le problème Π, on peut résoudre le problème Π'. Par contradiction, on montre que Π est au moins aussi difficile à résoudre que Π' (à la complexité de la réduction prés). Dans la plupart des cas on s'intéresse aux réductions polynomiales, c'est-à-dire celles demandant uniquement un espace et un temps polynomial pour être effectuées.

Pour montrer qu'un problème Π est **C**-difficile pour une classe **C** donnée, on montre qu'il existe une réduction entre un problème **C**-complet Π' et le problème Π.

1.4.3 Problème d'optimisation

Un problème d'optimisation est un problème algorithmique dans lequel on recherche une solution qui optimise la valeur d'un objet donné. De nombreux problèmes de décision peuvent être transformés en problèmes d'optimisation, la solution recherchée devient alors la meilleure solution réalisable. Une solution réalisable correspond à une solution trouvée respectant les contraintes imposées par le problème. Par exemple, considérons le problème du VOYAGEUR DE COMMERCE MINIMUM (voir ci-dessous). Dans ce problème, une solution réalisable est un chemin qui passe par tous les points exactement une fois et revient à son point de départ. Le problème étant de trouver parmi l'ensemble des solutions réalisables celle qui propose le chemin de longueur minimum. Cette solution est appelée solution optimale. Il est possible que plusieurs solutions soient optimales, l'ensemble de ces solutions est appelé ensemble de solutions optimales.

VOYAGEUR DE COMMERCE MINIMUM

Donnée : Soit n points (des "villes") et les distances séparant chaque point.
Solution : Un chemin de longueur k qui passe exactement une fois par chaque point et revienne au point de départ.
Mesure : k.

Pour tout problème d'optimisation, il existe un problème de décision correspondant qui demande s'il existe une solution réalisable pour une mesure donnée. Par exemple, le VOYAGEUR DE COMMERCE MINIMUM peut être transformé en problème de décision qui pose la question de l'existence d'un chemin passant par tous les sommets et de longueur inférieure à k (voir ci-dessous). Dans ce nouveau problème, k devient une donnée du problème.

VOYAGEUR DE COMMERCE

Donnée : Soit n points (des "villes"), les distances séparant chaque point et k un nombre réel supérieur à 0.
Question : Existe-t-il un chemin de longueur totale inférieur à k qui passe exactement une fois par chaque point et revienne au point de départ ?

Classe d'approximation

La classe **NPO** regroupe les problèmes d'optimisation dont la version décisionnelle appartient à la classe **NP**. Soit Π un problème d'optimisation, Π appartient à la classe **NPO** si les propositions suivantes sont respectées :

1. pour toute solution, il est possible de montrer que la solution est réalisable en temps polynomial,

2. la mesure de toute solution réalisable peut être évaluée en temps polynomial,

3. au moins une solution réalisable est calculable en temps polynomial.

Dans la présente thèse, nous nous intéressons seulement à des problèmes de la classe **NPO**.

En théorie de la complexité, l'approximation consiste à montrer qu'un problème d'optimisation peut être résolu par un algorithme A de complexité inférieure à l'algorithme résolvant le problème de décision correspondant à un écart près (entre la solution optimale et la solution trouvée par l'algorithme d'approximation A). Généralement, un algorithme d'approximation calcule en temps polynomial une solution réalisable (plus ou moins proche de la solution optimale) d'un problème dont le problème de décision correspondant n'est pas polynomial.

Par exemple, pour le problème du voyageur de commerce minimum, il a été montré qu'il existe un algorithme polynomial qui calcule un chemin (i.e. une solution réalisable) qui est, dans le pire des cas, $3/2$ fois plus long que le chemin le plus court (i.e. la solution optimale). Ce résultat est possible seulement sous l'hypothèse que l'inégalité triangulaire soit respectée entre les points (i.e. le chemin le plus court pour passer d'un point à un autre est toujours le chemin reliant les deux points, passer par un tierce point ne raccourcit jamais le chemin). On dit que le facteur d'approximation de cet algorithme est de $3/2$.

Noter que plus un facteur d'approximation est proche de 1, plus la solution trouvée est proche de la solution optimale et donc plus la solution trouvée est de qualité.

Nous considérons les classes suivantes :

- **PTAS** : pour schéma d'approximation en temps polynomial (de polynomial-time approximation scheme en anglais). Classe regroupant les problèmes **NPO** admettant un algorithme approché polynomial ayant un facteur d'approximation égal à $1 + \varepsilon$.

- **APX** : pour approximable. Classe regroupant les problèmes **NPO** admettant un algorithme approché polynomial ayant un facteur d'approximation constant.

- **Log-APX** : Classe regroupant les problèmes **NPO** admettant un algorithme approché polynomial ayant un facteur d'approximation logarithmique par rapport à l'entrée du problème.

- **Poly-APX** : Classe regroupant les problèmes **NPO** admettant un algorithme approché polynomial ayant un facteur d'approximation polynomial par rapport à l'entrée du problème.

Si un problème peut être approché par un facteur d'approximation donné, alors il peut également l'être par un facteur plus grand. De ce constat, il est possible de déduire que **PTAS** $\subseteq$ **APX** $\subseteq$**Log-APX** $\subseteq$ **Poly-APX**.

Remarquons que l'égalité entre deux classes restent un problème ouvert en informatique. Tout comme les classes de problème de décision, les classes de problème d'optimisation considèrent en plus les classes de type **C**-difficile et **C**-complet.

Réduction stricte et S-réduction

Afin de montrer l'appartenance à une classe **C**-difficile d'un problème d'optimisation, on utilise différentes réductions préservant le facteur d'approximation entre les problèmes d'optimisation. Il existe de nombreuses réductions préservant le facteur d'approximation (**L**-réduction, **APX**-réduction, etc), nous présentons seulement les réductions que nous utilisons dans le chapitre 5 : la réduction stricte et la **S**-réduction. Pour présenter ces réductions nous nous inspirons de la publication de Crescenzi [17] qui présente un panorama des différentes réductions existantes.

Considérons deux problèmes de la classe **NPO** Π et Π'. Pour tout problème $\Pi'' \in \{\Pi, \Pi'\}$, la notation $OPT_{\Pi''}$ correspond à la valeur de la solution optimale de Π'', $m_{\Pi''}(x'', y'')$ correspond au coût de la solution y'' de l'instance x'' du problème Π'' et $m_{\Pi''}(x'', y'')$ correspond à la mesure utilisée pour déterminer quelle solution est considérée comme optimale.

Étant donné une instance x d'un problème Π et une solution réalisable y de x, nous définissons le ratio de performance R_Π de y par rapport à x comme suit :

$$R_\Pi = max \left\{ \frac{m_\Pi(x, y)}{OPT(x)}, \frac{OPT(x)}{m_\Pi(x, y)} \right\}$$

Une stricte réduction est définie par deux fonctions exécutables en temps polynomial f et g telle que, pour chaque instance x de Π, $f(x)$ renvoie une instance de Π' et pour chaque solution réalisable y' de $f(x)$, $g(y')$ retourne une solution réalisable de x. Une réduction (f, g) de Π vers Π' est qualifiée de stricte si, pour toute instance x de Π et pour toute solution réalisable y' de Π', l'équation suivante est respectée :

$$R_\Pi(x, g(y')) \leq R_{\Pi'}(f(x), y').$$

Il existe une **S**-réduction de Π vers Π' si :

1. Pour toute instance x of Π, $OPT_{\Pi'}(f(x)) = OPT_\Pi(x)$.
2. Pour toute instance x de Π et pour toute solution réalisable y' de Π', $m_\Pi(x, g(y')) = m_{\Pi'}(f(x), y')$.

La **S**-réduction est un cas particulier de la réduction stricte, donc une **S**-réduction implique une réduction stricte. De plus, une **S**-réduction, préserve toutes les autres réductions préservant le facteur d'approximation [16]. Ainsi, en montrant qu'il existe une **S**-réduction d'un problème Π (appartenant à la classe **C**-difficile) vers un problème Π', on montre que Π' appartient à classe **C**-difficile.

II

Étude de cas : le Bitcoin

Bitcoin est le premier système à proposer un échange de valeurs directement de pair à pair sans passer par un service tiers. Bitcoin propose d'échanger de la crypto-monnaie (ou monnaie virtuelle) mais son système peut permettre d'échanger toute sorte de valeurs : argent, titre de propriété, crypto-monnaie, etc. La section suivante présente le fonctionnement du réseau Bitcoin.

2.1 Introduction

Bitcoin est présenté pour la première fois en novembre 2008 par une personne ou un groupe de personnes sous le pseudonyme de Satoshi Nakamoto [58] comme la première crypto-monnaie. La première transaction de Bitcoin est réalisée le 12 janvier 2009, depuis, Bitcoin est la crypto-monnaie la plus chère du marché et une des plus utilisées.

Le Bitcoin est une crypto-monnaie utilisant une blockchain publique. Elle propose un système d'échange de monnaie virtuelle (appelé bitcoin) décentralisé, autrement dit, Bitcoin propose un système bancaire sans banque ni tiers de confiance assurant le contrôle du réseau. Pour se faire, Bitcoin propose l'utilisation d'une blockchain distribuée dans un réseau pair-à-pair sécurisé par un consensus basé sur une preuve de travail que nous détaillons plus loin dans cette section.

Bitcoin est le premier réseau à proposer l'utilisation de la blockchain dans le but d'effectuer des transactions entre utilisateurs. Le consensus Bitcoin est basé sur un protocole de preuve de travail, que nous détaillons dans les sections suivantes, qui protège le réseau des attaques de déni de service (DoS) et les attaques par double dépense. Les blocs de la chaîne contiennent les transactions et donc l'ensemble des blocs constitue l'historique de l'ensemble des transactions effectuées sur le réseau depuis sa création. Ainsi de nouvelles transactions sont enregistrées lorsqu'un nouveau bloc est ajouté à la chaîne. Un nouveau

bloc est ajouté à la chaîne toute les dix minutes environ. Chaque bloc contient environ un millier de transactions, ce qui permet au réseau de supporter en moyenne six transactions par seconde. Bien que les comptes des utilisateurs ne sont pas totalement anonymes et que les transactions sont enregistrées en clair de façon permanente sur le registre, il est difficile de relier le solde d'un compte à une personne.

2.2 La chaîne bitcoin

Bitcoin propose un réseau d'échange de monnaie sans tiers de contrôle et donc sans banque qui centralise et/ou vérifie les transactions. Bitcoin utilise une blockchain qui enregistre l'historique des transactions depuis la création du réseau sans jamais les supprimer ou les modifier. Chaque utilisateur possède une copie de cet historique et le consensus permet à l'ensemble des utilisateurs de se mettre d'accord sur les transactions à rajouter pour qu'elles soient effectives. Il est donc nécessaire que les blocs soient structurés et formatés afin que l'ensemble des utilisateurs puisse se mettre d'accord sans ambiguïté sur leur validité.

Structure

Bitcoin propose un système d'échange de monnaie en enregistrant les transactions réalisées dans une blockchain. Pour qu'un bloc soit ajouté à la chaîne, il faut qu'un utilisateur le crée et le partage avec les autres membres du réseau, afin que ces derniers le valident et l'enregistrent à leur tour. Pour des soucis évidents de performances, la validation d'un bloc doit pouvoir se faire sans communication inter-sites. Bitcoin utilise donc un algorithme de consensus permettant à tous les utilisateurs de s'accorder sur l'état de la chaîne sans autre communication que le nouveau bloc à ajouter.

Pour qu'un tel accord soit possible, les blocs doivent répondre à un standard afin que l'ensemble des utilisateurs puisse vérifier si le bloc est bien valide et se prémunir des blocs incorrects ou contenant des transactions invalides. Les blocs se décomposent en deux parties : l'en-tête et les transactions (voir figure 2.1).

Les transactions sont enregistrées sous la forme de liste d'UTXOs (voir section 2.3). L'en-tête regroupe l'ensemble des informations d'un bloc (mises à part les transactions). Il permet notamment de réaliser la preuve de travail que nous verrons dans la section suivante. La taille de l'entête est toujours de 80 octets. Un en-tête (voir exemple figure 2.2) est composé des données suivantes :

– **Version (4 octets) :** la version du protocole Bitcoin utilisé pour le bloc. L'historique des versions est disponible sur le site bitcoin.org [1].

1. https://bitcoin.org/en/version-history

FIGURE 2.1 – *Structure des blocs de la chaîne du réseau Bitcoin.*

– **Hash de l'en-tête précédent (32 octets) :** le résultat du hachage de l'en-tête du bloc précédent par la fonction SHA-256 inversé octet par octet. Il permet de lier les blocs entre eux, de rendre la chaîne non modifiable dans le temps.

– **Racine de Merkle (32 octets) :** la racine de l'arbre de Merkle inversé octet par octet, construit à partir des transactions contenues dans le bloc. Le hash de chaque transaction correspond à une feuille de l'arbre. Les hashs obtenus sont concaténés deux à deux et re-hachés jusqu'à ce qu'il n'en reste plus qu'un : la racine de l'arbre de Merkle. La fonction de hachage utilisée est SHA-256. La racine de Merkle permet de figer les transactions du bloc. En effet, si l'une d'entre elles est modifiée, la racine de Merkle sera également modifiée, ce qui engendre une modification de la valeur du hash de l'en-tête du bloc et, puisque l'en-tête du bloc suivant contient le hash de cet en-tête, l'ensemble des blocs suivants sera invalidé.

– **Horodatage (4 octets) :** date de création du bloc en heure Unix (nombre de secondes écoulées depuis le 1er Janvier 1970 00:00:00 UTC). L'horodatage est approximatif, il est défini par le site qui crée l'en-tête. Pour qu'un horodatage soit validé par un site, il doit respecter les deux contraintes suivantes : l'horodatage doit être antérieur à deux heures dans le futur par rapport à l'heure de la machine (contrainte d'antériorité) et postérieur à l'horodatage médian des onze derniers blocs (contrainte de postériorité). La contrainte d'antériorité est utilisée pour éviter que la chaîne ne soit trop loin dans le temps par rapport à la date réelle. La contrainte de postériorité prenant en compte l'horodatage médian des onze derniers blocs (et pas seulement l'horodatage du dernier) ne garantit pas que l'horodatage des blocs soit chronologique (un bloc n'a pas nécessairement un horodatage postérieur au bloc qui le précède). En revanche, la contrainte de postériorité permet de résoudre le problème qui

Champs :	Exemple :
Version	02000000
Hash de l'en-ête du bloc précédent	b6ff0b1b1680a2862a30ca44d346d9e8
inversé octet par octet	910d334beb48ca0c0000000000000000
Racine de l'arbre de Merkle	9d10aa52ee949386ca9385695f04ede2
inversé octet par octet	70dda20810decd12bc9b048aaab31471
Horodatage	24d95a54
Bits	30c31b18
Nonce	fe9f0864

FIGURE 2.2 – *Exemple d'en-tête d'un bloc Bitcoin.*

apparaît lorsqu'un bloc est validé avec un horodatage de deux heures dans le futur. En effet, considérant un site qui enregistre un bloc avec un horodatage de deux heures dans le futur (par rapport à l'heure du site), si les blocs doivent être enregistrés par horodatage chronologique et en considérant la contrainte d'antériorité, le bloc suivant devrait avoir un horodatage à la fois antérieur et postérieur à deux heures dans le futur. Dans ce cas là, le site devrait attendre quelques secondes après l'enregistrement du bloc pour que son heure avance afin de repousser la contrainte d'antériorité, lui donnant ainsi un petit domaine de valeurs pour l'horodatage du prochain bloc. Notez que les sites sont généralement d'accord sur l'heure actuelle (notamment grâce au Network Time Protocol[2]) et qu'ils ont accès aux blocs précédents. Ils peuvent donc facilement vérifier que les en-têtes qu'ils reçoivent respectent bien les contraintes d'antériorité et de postériorité.

– **Bits (4 octets) :** valeur permettant de calculer la valeur du hash cible. Le hash cible ou cible (target en anglais) est un nombre binaire de 256 chiffres. Pour qu'un bloc soit valide, il faut que le hash de son en-tête soit inférieur ou égal à la valeur de la cible. Le Bits est une variable globale du réseau, connu par l'intégralité de ses acteurs et elle est recalculée tous les 2016 blocs. Le premier octet du Bits correspond à la valeur de l'exposant de la cible et les 3 derniers octets correspondent aux 3 premiers octets de la cible. Considérons $O_1, \ldots, O_4$ les quatre octets composant le Bits B avec O_i l'octet en position i dans B. Alors on peut calculer la cible C (en hexadécimal) avec la formule suivante : $C = (O_2 O_3 O_4)^{O_1}$. Un exemple de conversion est présenté figure 2.6.

– **Nonce (4 octets) :** nombre binaire de 32 chiffres. Lorsqu'un site crée un bloc, il fait varier le nonce jusqu'à ce que le hash de l'en-tête soit inférieur à la cible. Le nonce permet de prouver que le travail nécessaire pour ajouter un bloc à été réalisé. Plus de détails dans la section 2.4.

Les blocs sont également bornés en taille, un bloc ne peut pas faire plus de 1 Mo. L'en-tête mesurant 80 octets, la taille totale des transactions ne doit pas excéder 999 920 octets.

2. https://en.wikipedia.org/wiki/Network_Time_Protocol

Différents types d'acteurs

Il existe différents types de nœuds (i.e. différents acteurs) sur le réseau Bitcoin qui, ensemble, garantissent le bon fonctionnement de la blockchain et, grâce à un consensus, enregistrent et vérifient les transactions des nouveaux blocs afin que la chaîne et ses acteurs restent à jour. Le réseau Bitcoin étant un réseau entièrement public, n'importe quel utilisateur peut choisir de jouer le rôle qu'il souhaite. Il existe trois types d'utilisateurs différents sur le réseau Bitcoin (et sur la plupart des blockchains publiques) :

- **Nœud complet :** (en anglais, full node) sont des sites possédant l'intégralité de la blockchain (du bloc genèse au dernier bloc en date) et qui la tiennent à jour régulièrement. Chaque nœud complet est vérifié, authentifié et stocké par tous les autres nœuds complets. Les nœuds complets sont les garants du bon fonctionnement de la blockchain, en validant une à une les transactions et les blocs qu'ils reçoivent puis en les relayant aux autres nœuds complets. Les nœuds complets doivent avoir une puissance de calcul plus importante qu'un utilisateur classique ainsi qu'une mémoire suffisante pour enregistrer l'entièreté de la chaîne (environ 320 Go en Février 2021) et les transactions en cours. Actuellement, on estime qu'il y a environ 10 000 nœuds complets sur le réseau Bitcoin. La force du réseau Bitcoin réside dans le fait que chacun peut devenir, s'il le souhaite, un nœud complet du réseau.
- **Nœud léger :** (en anglais, light nodes) sont des sites qui ne possèdent pas l'intégralité de la blockchain mais seulement les en-têtes qui les intéressent. Les nœuds légers sont connectés à des nœuds complets et peuvent vérifier que certaines transactions ont été enregistrées dans un bloc spécifique grâce notamment à la racine de Merkle et au protocole de vérification de paiement simplifié (Simplified Payment Verification, SPV, défini dans l'article de Nakamoto [58], voir figure 2.3). Contrairement aux nœuds complets, les nœuds légers ne stockent pas une copie de l'historique complet du réseau et font confiance aux nœuds complets pour leur fournir des données validées. Les nœuds légers permettent de décentraliser un peu plus le réseau. Ils traitent moins de données que les nœuds complets et nécessitent beaucoup moins de ressources (un ordinateur de bureau suffit). Généralement, les utilisateurs du réseau Bitcoin qui ne possèdent pas la chaîne (i.e. des utilisateurs lambda qui veulent utiliser des bitcoins sans participer au réseau), se connectent à des nœuds légers pour qu'ils vérifient que les transactions qui les concernent ont été réalisées. Le fonctionnement des nœuds légers permet d'alléger le travail des nœuds complets en faisant le lien entre eux et les utilisateurs qui veulent utiliser des bitcoins sans être acteur sur le réseau. Les nœuds légers permettent donc à la blockchain de se développer plus durablement que si elle ne possédait que des nœuds complets.
- **Mineur :** (en anglais, miner) site qui construit et partage les blocs à ajouter à la chaîne. Les mineurs sont récompensés pour calculer les preuves de travail dans le but de construire des blocs valides et de les transmettre aux nœuds complets qui les vérifient et les enregistrent à la chaîne s'ils sont valides. Nous abordons plus profondément le rôle des mineurs dans la section 2.4. La difficulté de la preuve de travail implique qu'un mineur a besoin d'une puissance de calcul conséquente s'il veut réussir à proposer un nouveau bloc à la chaîne.

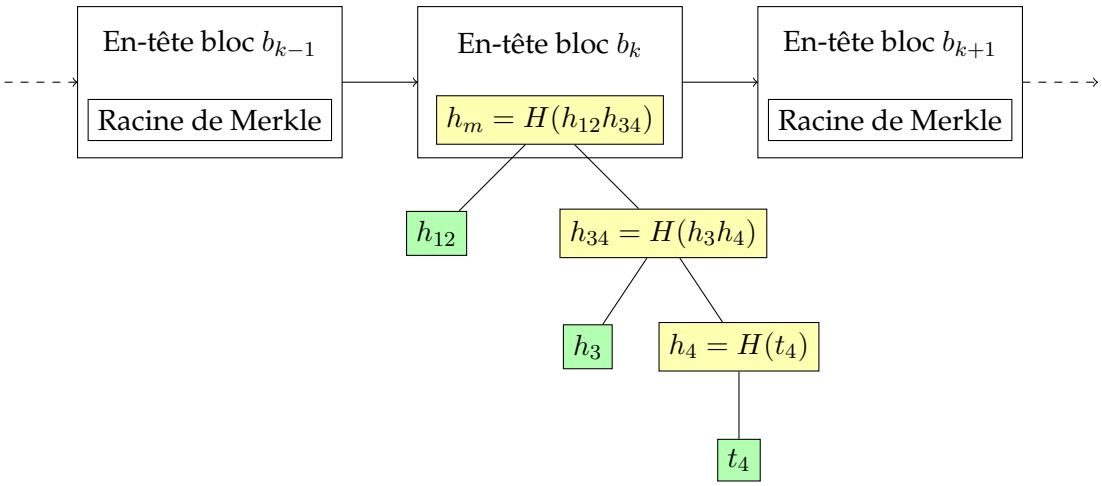


FIGURE 2.3 – *Exemple de fonctionnement du protocole de vérification de payement simplifié (Simplified Payment Verification, SPV). Dans cet exemple, le nœud léger possède les en-têtes des blocs et veut vérifier que la transaction t_4 à bien été enregistrée dans le bloc b_k. Pour ce faire, il récupère les hashs h_3 et h_{12} (en vert) grâce à un nœud complet. Ensuite, il recalcule les hashs h_4, h_{34} et h_m (en jaune). Si la racine de Merkle du bloc b_k est égale à h_m alors b_k contient la transaction t_4. Le payement que réalise la transaction t_4 a bien été enregistré dans la chaîne, il a été effectué.*

La figure 2.4 illustre la connexion entre les différents acteurs du réseau, le table 2.1 résume les pouvoirs de chaque acteur.

	Proposer un nouveau bloc	Initier une transaction	Historique complet de la chaîne
Nœuds complets	Non	Oui	Oui
Nœuds légers	Non	Oui	Non
Mineurs	Oui	Non	Non

TABLE 2.1 – *Résumé des rôles des acteurs sur le réseau Bitcoin.*

2.3 Les transactions

Bitcoin propose à ses utilisateurs d'effectuer des transactions de pair à pair sans passer par un tiers de contrôle. Pour ce faire, la blockchain de bitcoin enregistre l'ensemble des transactions réalisées depuis la création du réseau, constituant ainsi l'historique de toutes les transactions du réseau. Dans cette section, nous présentons le modèle utilisé par Bitcoin afin de standardiser et d'enregistrer les transactions dans la chaîne et comment le réseau récompense les mineurs pour les inciter à enregistrer ces transactions.

Le modèle utilisant des UTXOs (pour unspent transaction outputs en anglais ou sorties de transactions non dépensées en français) utilisé par Bitcoin est un modèle introduit par Nakamoto [58] en 2008.

Une UTXO est une sortie de transaction qui n'a pas encore été dépensée (e.g. utilisée pour l'entrée d'une transaction ultérieure). Techniquement, une UTXO correspond

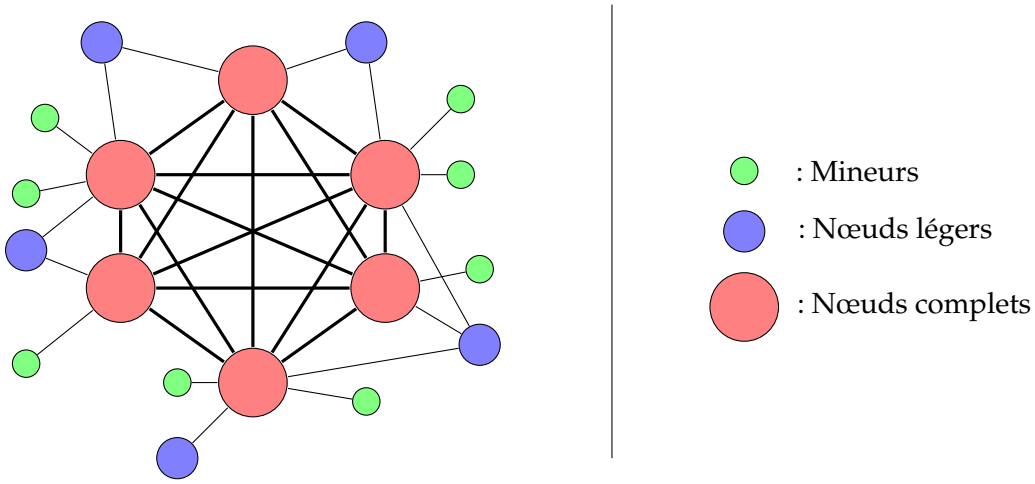


FIGURE 2.4 – *Exemple d'organisation du réseau Bitcoin, les nœuds rouges représentent les nœuds complets, les bleus les nœuds légers et les verts les mineurs. Un trait relit deux acteurs quand ils sont connectés entre eux et qu'ils peuvent communiquer.*

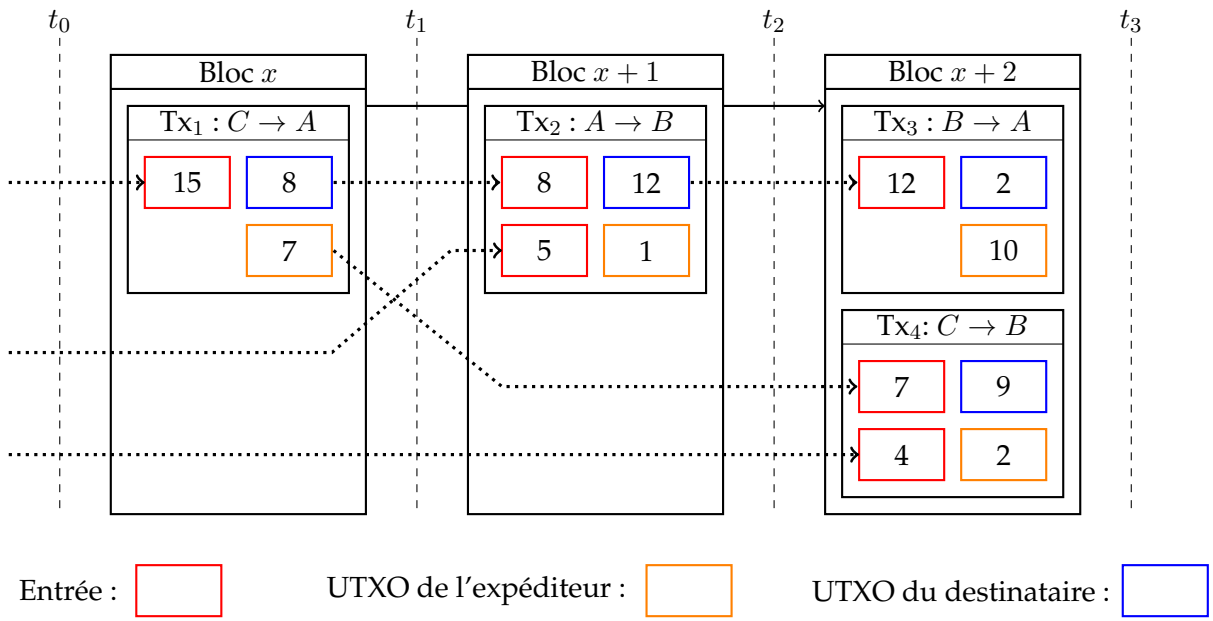
à une adresse où se trouve une sortie de transaction à laquelle est associée une somme correspondant à la somme disponible sur cette UTXO. Par abus de langage on associe directement une UTXO à une somme. La somme des UTXOs d'un utilisateur constitue son solde et la somme de l'ensemble des UTXOs de la chaîne correspond à la somme totale de monnaie disponible pour le réseau.

Afin de vérifier l'identité de l'initiateur d'une transaction, chaque transaction est signée par un système de cryptographie asymétrique (voir sous-section 1.2.2). Chaque utilisateur possède une clef publique et une clef privée générées à partir de courbes elliptiques [56].

Lorsqu'un utilisateur veut réaliser une transaction, il doit référencer en entrée une ou plusieurs transactions passées dont la somme totale des sorties est supérieure ou égale au montant de la transaction qu'il souhaite réaliser. Plus précisément, quand un utilisateur veut réaliser une transaction, il doit constituer d'une part un ensemble d'UTXOs lui appartenant et dont la somme est supérieure ou égale à la somme qu'il veut transférer et d'autre part un ensemble d'adresse d'arrivée associé à une somme qui correspondront aux nouveaux UTXOs. Une fois que la transaction est créée et signée, l'utilisateur la transmet au reste du réseau et attend qu'elle soit validée et enregistrée sur la chaîne. Une transaction est considérée comme valide si la signature correspond à l'envoyeur et si les UTXOs renseignées en entrée ne sont pas déjà utilisées. Une fois la transaction enregistrée, les UTXOs utilisées en entrée sont "consommées" (elles sont encore sur la chaîne mais ne pourront plus jamais être utilisées pour une autre transaction). Après chaque transaction enregistrée dans la chaîne, chaque utilisateur met à jour l'ensemble des UTXOs disponibles, les UTXOs utilisées en entrée de transaction sont supprimées tandis que celles en sortie sont ajoutées.

Prenons un exemple pour clarifier le fonctionnement des transaction et l'utilisation des UTXOs (l'exemple suivant correspond à la transaction Tx_2 de la figure 2.5).

Considérons Alice, une utilisatrice de Bitcoin possédant 28 bitcoins répartis sur quatre UTXOs comme suit :



Temps	t_0	t_1	t_2	t_3
UTXOs de A	{10, 5, 5}	{10, 8, 5, 5,}	{10, 5, 1}	{10, 5, 2, 1}
UTXOs de B	{}	{}	{12}	{10, 9}
UTXOs de C	{15, 4}	{7, 4}	{7, 4}	{2}

FIGURE 2.5 – *Exemple de fonctionnement du modèle UTXO. Les entrées sont des UTXO existant, appartenant à l'expéditeur avant la transaction. Après la transaction, les UTXO utilisées en entrée sont détruites et les UTXO créées pour l'expéditeur et le destinataire deviennent disponibles pour de futures transactions.*

- un de 10 bitcoins,
- un de 8 bitcoins,
- deux de 5 bitcoins.

Alice désire envoyer 12 bitcoins à Bob, un nouvel utilisateur de la blockchain (qui ne possède pas encore de bitcoins, et donc pas d'UTXO). Il n'est pas possible d'utiliser seulement une partie d'une UTXO pour réaliser une transaction. Alice doit donc combiner ses UTXOs afin que leur somme soit au moins égal à 12. Dans notre exemple, Alice choisit de combiner son UTXOs de 8 bitcoins avec un à 5. Ainsi, elle peut envoyer les 13 bitcoins vers deux nouvelles UTXOs, une de 12 bitcoins appartenant à Bob et une de 1 bitcoin appartenant à Alice. Les deux UTXOs utilisées en entrée deviennent inutilisables, elles ne sont plus considérées comme des UTXOs et ne pourront donc pas être utilisées pour une transaction ultérieure. Après que la transaction soit enregistrée sur la blockchain, Alice possède 16 bitcoins répartis sur trois UTXOs :

- une de 10 bitcoins,
- une de 5 bitcoins,
- une de 1 bitcoin,

et Bob possède une seule UTXO de 12 bitcoins.

À noter que lorsqu'un utilisateur initie une transaction, il faut qu'il paye des frais (d'un montant qu'il fixe lui-même) au site qui l'enregistrera sur la chaîne. Étant donné qu'un site choisit les transactions qu'il veut pour former le bloc à enregistrer, il va avoir tendance à choisir les transactions qui proposent de payer les frais les plus importants et donc, plus une transaction fixe des frais élevés, plus elle a de chance d'être enregistrée rapidement. Techniquement, les frais sont égaux à la différence entre les UTXOs utilisées en entrée et celles utilisées en sortie. Une UTXO appartenant au site qui enregistre le bloc est ajoutée en sortie afin d'équilibrer la somme des entrées et des sorties.

Comme nous le verrons plus tard, l'utilisateur qui enregistre le bloc dans la chaîne est également récompensé en gagnant un certain nombre x de bitcoins. Cette récompense prend la forme d'une transaction spéciale dans laquelle il n'y a pas d'envoyeur mais seulement un receveur : l'utilisateur qui ajoute le bloc. Ainsi cet utilisateur gagne un UTXO de x bitcoins.

La taille totale des transactions ne doit pas dépasser 999 920 octets. Les transactions n'étant pas de taille constante (suivant le nombre d'UTXOs en entrée et en sortie), il est difficile de dire précisément combien de transactions peuvent contenir un bloc. La communauté Bitcoin estime qu'une transaction mesure en moyenne 250 octets et un bloc est ajouté toutes les dix minutes (600 secondes) en moyenne. On peut donc calculer le nombre maximal de transactions par seconde (TPS) : TPS $= 999\,920 \,/\, 250 \,/\, 600 \simeq 6,6$ transactions par seconde.

2.4 Consensus

2.4.1 Preuve de travail

Dans le consensus Bitcoin, les mineurs travaillent en concurrence pour calculer une preuve de travail [23, 41] (i.e. résoudre un problème mathématique asymétrique), voir sous-section 1.3.4. Le premier mineur à trouver une preuve de travail valide peut construire, ajouter et partager le bloc suivant. Ainsi, ce mineur aura construit et enregistré un bloc et il remportera la récompense et les frais de transactions de ce bloc.

Le problème mathématique utilisé pour preuve de travail du réseau Bitcoin consiste à construire un bloc (en respectant le format figure 2.1) tel que le hash de son en-tête résultant de la fonction de hachage SHA-256 soit inférieur à une valeur cible (calculée à partir de la variable "Bits" de l'en-tête). Le mineur construit donc un ensemble de transactions valides, ainsi que l'en-tête du bloc, puis fait varier le nonce jusqu'à obtenir un en-tête tel que son hash est inférieur à la cible. Le nonce trouvé sera alors qualifié de valide. À noter que le nonce doit rester un entier de 4 octets, si un mineur à testé toutes les valeurs possibles (soit 2^{32} possibilités), il peut également modifier la valeur de l'horodatage (à condition que la nouvelle date fixée respecte les contraintes sur l'horodatage) et ainsi continuer à chercher un nonce valide.

Bits : 0x 18 06 96 F4

Cible : 0x 06 96 F4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FIGURE 2.6 – *Conversion entre le Bits et la cible. Le premier octets du Bits (en rouge) correspond à l'exposant de la cible et les trois autres octets (en bleu) sont les trois premiers octet de la cible. Dans notre exemple, 0x18 = 24 en base 10, donc la cible sera composée de 24 octets (soit 48 caractères), les trois premiers égaux aux trois derniers du Bits, le reste est complété avec des 0.*

Étant donné que la fonction SHA-256 est une fonction de hachage considérée comme solide, un mineur ne peut pas construire un bloc à partir d'un hash. Il n'a donc pas de meilleure solution que de tester des nonces jusqu'à en trouver un valide. Il ne peut ni prévoir, ni estimer la valeur que devra prendre ce nonce. Ainsi, il n'existe pas d'algorithme plus efficace que l'algorithme de force brute qui teste tous les nonces possibles. Cette propriété implique la capacité d'un mineur à trouver un nonce valide dépend directement de sa puissance de calcul. Autrement dit, si un mineur a dix fois plus de puissance de calcul qu'un autre, il calculera dix fois plus de hash que lui, il aura donc dix fois plus de chance de trouver un nonce valide et donc il rajoutera en moyenne dix fois plus de blocs que lui.

2.4.2 Validation de blocs

Quand un mineur trouve un nonce tel que le hash de l'en-tête du bloc qu'il a créé est inférieur à la cible, il peut partager le bloc aux nœuds complets. Lorsqu'un nœud complet reçoit un bloc, il doit vérifier sa validité avant de l'enregistrer à sa propre chaîne. Pour qu'un bloc soit considéré comme valide par un nœud complet, le bloc doit respecter les points suivants :

- La version du consensus utilisée correspond à la version actuelle.
- Le hash du bloc proposé correspond bien au hash de l'en-tête du bloc précédent.
- Les transactions enregistrées dans le bloc sont valides, elles sont toutes signées par l'émetteur de la transaction et aucune UTXO d'entre elle n'a déjà été utilisée.
- La racine de l'arbre de Merkle de l'en-tête correspond bien à la racine obtenue à partir des transactions du bloc.
- L'horodatage respecte bien les contraintes d'antériorité à deux heures dans le futur et de postériorité à l'horodatage médian des 11 derniers blocs.
- La valeur du Bits utilisée est la valeur correspondante au Bits actuel du réseau.
- Le résultat de la fonction de hachage SHA-256 du nonce proposé, concaténé au reste de l'en-tête, est inférieur à la cible obtenue à partir du Bits.

Quand un nœud complet valide un bloc, il l'ajoute à sa chaîne et le transmets à ses voisins. Si le nœud était un mineur, il arrête de calculer le nonce du bloc qu'il tentait d'ajouter et il crée un nouveau bloc pour tenter de l'enregistrer à la suite du bloc qu'il vient de valider.

2.4.3 Stabilisation de la chaîne

Fork

Dans cette partie, nous ne considérerons pas les forks volontaires de la chaîne Bitcoin (qui apparaissent quand des utilisateurs du réseau veulent modifier le consensus).

Étant donné que tous les mineurs du réseau essayent de rajouter un bloc différent sur la chaîne en même temps, il est possible que deux d'entre eux (ou plus) enregistrent et partagent un bloc en même temps. Dans ce cas là, les nœuds complets vont recevoir deux blocs, tous les deux valides et ne pourront pas choisir lequel ajouter à la chaîne (généralement ils ajoutent le premier qu'ils reçoivent). Or, étant donné que la communication inter-sites est asynchrone, il est très probable que tous les nœuds complets ne reçoivent pas les deux blocs dans le même ordre. Dans ce cas là, le réseau se retrouve coupé en deux (chaque partie travaillant sur le bloc qu'elle a choisi de valider) et donc la blockchain n'est plus une chaîne. Pour prémunir le réseau de ces forks, le consensus Bitcoin propose de travailler toujours sur la chaîne la plus longue.

Considérons b_3 le dernier bloc de la chaîne, b_4 et b'_4 deux blocs trouvés dans le même laps de temps à la suite de b_3 (voir figure 1.9). Lorsqu'un fork involontaire apparaît, chaque nœud peut considèrer le bloc qu'il souhaite entre b_4 et b'_4) dans un premier temps. Lorsqu'un autre bloc valide est créé (par exemple b_5 à la suite du bloc b_4), il est transmis à l'ensemble des nœuds complets. Ainsi, puisque la chaîne contenant b_5 et b_4 est plus longue que celle qui contient b'_4 et en accord avec la règle de travailler toujours sur la chaîne la plus longue, tous les nœuds complets considérerons b_4 et b_5 valide. Le bloc b'_4 est abandonné, il n'appartient plus à la chaîne, il devient donc orphelin et les transactions qu'il contenait ne sont donc pas enregistrées (mis à part si elles sont présentes dans b_4 ou b_5), mais pourront toujours l'être dans un bloc futur.

Imaginons qu'un bloc b'_5 succédant à b'_4 soit trouvé dans le même laps de temps que b_5, alors le problème est reporté au bloc suivant. Étant donné que le temps de calcul d'un bloc (dix minutes en moyenne) est beaucoup plus élevé que son temps de propagation (quelques secondes), les forks sont très rares (qu'il se prolonge sur un autre bloc à la suite encore plus) et donc la chaîne sera très rapidement stabilisée.

Rappelons que plus un mineur a une puissance de calcul élevée, plus il a de chances de trouver le prochain bloc. Donc, en cas de fork, un bloc aura autant de chance d'être enregistré définitivement que le pourcentage de puissance de calcul qui le considère sur le réseau. Par exemple, reprenons la figure 1.9, le bloc b_3 est enregistré et il apparaît un fork quand les mineurs tentent d'ajouter le bloc suivant. Deux mineurs créent et envoient dans le même laps de temps les blocs b_4 et b'_4 aux nœuds complets. Considérons également que le bloc b_4 est partagé un peu plus vite que b'_4 (si b_4 est trouvé un peu avant b'_4 ou à cause de la disposition géographique des nœuds par exemple). Supposons que 70% de la puissance de calcul du réseau reçoivent b_4 en premier et que les 30% restants reçoivent b'_4 en premier. Dans ce cas là, 70% de la puissance de calcul du réseau essaie d'ajouter un bloc à la suite de b_4 et 30% à la suite de b'_4, on peut donc considérer que b_4 et b'_4 ont respectivement 70% et 30% de chance d'être enregistrés définitivement dans la chaîne et donc respectivement

30% et 70% de chance de devenir un bloc orphelin.

Équilibre et évolution dans le temps

Le réseau Bitcoin à été pensé pour qu'un bloc soit ajouté toutes les dix minutes en moyenne. Or, comme présenté dans la section précédente, le temps mis par un mineur pour effectuer le travail demandé dépend directement de sa puissance de calcul (plus vite il calcule des hashs plus vite il trouvera un bloc valide à enregistrer). Autrement dit, plus la puissance de calcul mise à disposition du réseau est importante, plus les blocs vont être enregistrés rapidement (et inversement). Cette dernière phrase serait vrai si la cible des blocs n'était pas recalculée régulièrement.

En effet, la difficulté des mineurs à trouver un bloc valide dépend directement de la cible. Plus la cible est élevée, plus il est facile de trouver un bloc tel que le hash de son en-tête lui soit inférieur. La cible est recalculée tous les 2016 blocs (soit quatorze jours si on considère un bloc toutes les dix minutes en moyenne) afin de garder une moyenne de dix minutes entre chaque bloc peu importe la puissance de calcul du réseau.

Soit k un entier strictement positif et multiple de 2016. Considérons C_k la valeur de la cible du bloc k, h_k et h_{k-2016} les horodatages des blocs k et $k-2016$ respectivement en heure Unix (nombre de secondes écoulées depuis le 1^{er} janvier 1970 00:00:00). Afin de calculer la cible C_{k+1} du bloc $k+1$, on calcule le ratio r correspondant au ratio entre le temps mis par le réseau pour enregistrer les 2016 derniers blocs et le temps qu'il aurait dû mettre si tous les blocs étaient espacés de dix minutes exactement (soit deux semaines ou 1 209 600 secondes) :

$$r = \frac{h_{k-2016} - h_k}{1\ 209\ 600}$$

Ensuite l'ancienne cible C_k est multipliée par r. La cible du bloc suivant C_{k+1} correspondra au résultat de $C_k \times r$ duquel on ne garde que les trois octets de poids fort, tous les autres sont mis à 0 (i.e. on ne prend en compte que les trois premiers octets du résultat, les autres passent à 0) afin d'enregistrer la cible avec seulement 4 octets (le Bits). À noter que r doit être compris entre 0,25 et 4 afin d'éviter que la nouvelle cible soit trop éloignée de la précédente (pas plus de 4 fois plus grande ou 4 fois plus petite). La figure 2.7 présente un exemple d'ajustement de la cible au bloc 6048.

Point important à constater, tous les acteurs du réseau ont accès à toutes les informations nécessaires pour calculer la cible. Ainsi, tous les acteurs peuvent calculer/vérifier la cible de n'importe quel bloc sans jamais communiquer et donc sans ralentir le réseau.

Comme présenté dans la section 2.3, x bitcoins sont gagnés par le mineur qui parvient à ajouter un bloc à la chaîne. Ainsi, à chaque fois qu'un bloc est enregistré sur la chaîne, x nouveaux bitcoins sont injectés sur le réseau. Toutefois, le bitcoin n'est pas une monnaie inflationniste, bien au contraire, le nombre de bitcoins est borné car la valeur de x décroît au fil du temps. En effet, la récompense donnée au mineur qui enregistre le bloc (x) est divisée tous les 210 000 blocs (environs quatre ans pour une moyenne de dix minutes par blocs) et après 64 divisions, les blocs ne généreront plus de bitcoin ($x = 0$). Toutefois,

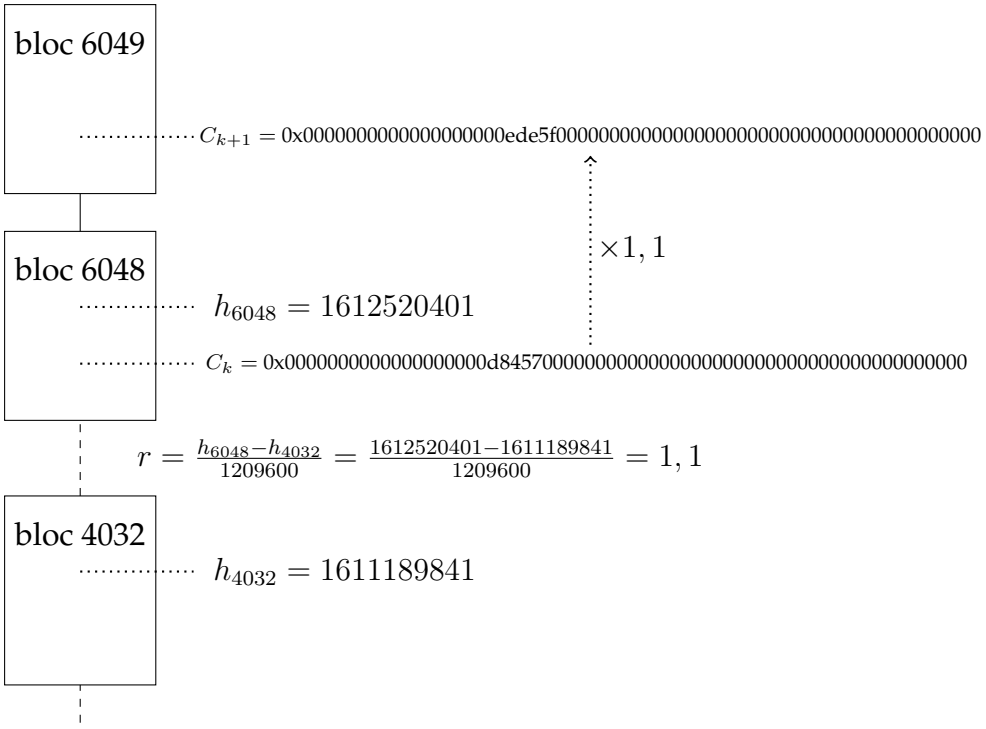


FIGURE 2.7 – *Exemple de calcul de la cible pour le bloc numéro 6049. Le ratio r est calculé à partir des horodatages des blocs 4032 et 6048. La cible du bloc 6048, C_k est ensuite multipliée par r et on ne conserve que les trois premiers octets de la nouvelle cible C_{k+1}, les autres octets sont mis à 0.*

les mineurs continueront d'être récompensés par le biais des frais de transactions. Au départ, en 2009, chaque bloc enregistré récompensait le mineur de 50 bitcoins ($x = 50$). Actuellement, le mineur qui enregistre un bloc gagne 12,5 bitcoins (environ 370 000€ [3]). À l'heure actuelle plus de 18 600 000 bitcoins ont déjà été générés. Les derniers devraient être injectés autour de l'an 2140 pour atteindre un nombre total de 21 000 000 de bitcoins.

2.5 Critiques

Le réseau Bitcoin à pour principal avantage d'être extrêmement bien sécurisé et résistant aux pannes, toutefois il présente également de nombreux défauts tels que sa consommation d'énergie élevée ou encore sa lenteur. Nous présentons dans cette section les principaux avantages et inconvénients du réseau Bitcoin.

2.5.1 Sécurité

La sécurité est le point fort du réseau Bitcoin. Comme vu précédemment, les nœuds complets sont ceux qui vérifient et enregistrent la chaîne, ce sont donc eux qui seront ciblés en cas d'attaque.

Les attaques par déni de service (en anglais Denial of Service, Dos) ou par déni de

3. Selon le cours du 29/01/2021 (https://coinmarketcap.com/)

service distribué (en anglais Distributed Denial of Service, DDos) [4] sont des attaques qui consistent à inonder un service de requêtes jusqu'à ce qu'il ne soit plus en capacité de répondre et donc d'empêcher les utilisateurs légitimes de ce service de l'utiliser. La décentralisation du réseau Bitcoin ainsi que la preuve de travail et le temps de calcul asymétrique qu'elle propose permet au système d'être totalement protégé contre les attaques DoS/DDoS. La preuve de travail demandée étant si longue à être calculée qu'il est impossible pour un ou plusieurs attaquants d'inonder le réseau avec des blocs valides, en revanche, ils peuvent tenter d'envoyer autant de blocs non valides qu'ils le souhaitent. Toutefois, lorsqu'un nœud complet reçoit un bloc non valide, il peut le traiter presque instantanément, il lui suffit de recalculer son hash. Il paraît donc difficile de bloquer un nœud complet en lui envoyant des requêtes (surtout qu'il pourrait rapidement choisir de filtrer ses messages après x blocs invalides envoyés par un même utilisateur). De plus, bloquer un nœud complet n'affecterait que très peu le réseau. Pour le bloquer totalement, les attaquants doivent bloquer l'intégralité des nœuds complets, soit un peu plus de 10 000 nœuds en moyenne sur l'année dernière [5].

Le réseau Bitcoin propose également une grande sécurité contre les attaques de double dépense (voir section 1.1.3). Supposons que les nœuds complets valident seulement les transactions valides (c'est-à-dire seulement les transactions bien signées et telles que le compte à débiter possède un solde suffisant pour réaliser la transaction). Considérons t_1 et t_2 deux transactions prenant en entrée les mêmes UTXOs mais des UTXOs différents en sortie. Une double dépense sur le réseau Bitcoin peut être réalisée de la façon suivante (les étapes sont illustrées une à une dans la figure 2.8) :

1. L'attaquant réalise une transaction t_1 pour utiliser une première fois ses bitcoins, il peut par exemple les utiliser pour acheter une voiture. Cette transaction est envoyée aux mineurs qui l'incluront dans un bloc b_k que les nœuds complets enregistreront à la chaîne. À cette étape il s'agit encore d'une transaction classique, valide, qui est enregistrée et validée par l'ensemble des nœuds complets du réseau. Une fois la transaction t_1 validée, l'attaquant récupère son gain, dans notre exemple la voiture.

2. En parallèle, l'attaquant crée localement un fork au niveau d'un bloc antérieur à b_k puis calcule la preuve de travail des blocs suivants en incluant la transaction t_2 jusqu'à obtenir une chaîne plus grande que la chaîne actuelle possédée par les nœuds complets.

3. L'attaquant envoie la chaîne qu'il a calculé aux nœuds complets qui considéreront alors que la chaîne légitime est celle de l'attaquant (car elle est plus grande que la leur). Ainsi, l'ensemble des blocs se trouvant après le fork (dont b_k) deviennent orphelins et l'ensemble des transactions qu'ils contenaient (dont t_1) deviennent invalidées. Lorsque la transaction t_1 devient invalidée, elle est renvoyée aux mineurs mais ils ne la considéreront plus comme valide, étant donné que les UTXOs en entrée ont déjà été utilisées par la transaction t_2. Si les UTXOs en sortie de t_2 appartiennent à l'attaquant, alors il a obtenu son gain (la voiture) sans avoir perdu ses bitcoins qu'il pourra redépenser.

4. https://en.wikipedia.org/wiki/Denial-of-service_attack
5. https://bitnodes.io/

Le problème de la double dépense dans une blockchain ne considérant que la chaîne la plus longue (ce qui est le cas de Bitcoin et de la plupart des blockchains) est fortement lié à la capacité d'écriture de l'attaquant sur la chaîne. En effet, si un utilisateur malveillant (ou un ensemble d'utilisateurs malveillants) rassemble plus de 50% de la capacité d'écriture du réseau, il peut écrire sur la blockchain plus rapidement que le reste du réseau. L'attaquant peut donc créer un fork et réécrire la chaîne depuis un bloc passé jusqu'à obtenir une chaîne plus longue que la chaîne courante qui sera donc remplacée. Dans ce cas, tous les blocs qui ont été calculés depuis le fork deviennent orphelins et toutes les transactions qu'ils contenaient sont invalidées.

Bitcoin est protégé des attaques par double dépense grâce à la preuve de travail. En effet, comme décrit précédemment, l'attaquant doit calculer des blocs jusqu'à obtenir une chaîne plus grande que la chaîne actuelle. Tant que la fonction de hachage SHA-256 reste solide, l'attaquant doit posséder une puissance de calcul supérieure au reste du réseau (d'où l'autre nom donné à cette attaque : l'attaque des 51%) s'il veut être certain que l'attaque aboutisse. Généralement, une transaction est considérée comme sure (i.e. le bloc qui la contient ne sera pas victime d'une attaque de double dépense) lorsque six blocs la succèdent. Donc si le vendeur d'un objet reçoit un paiement en Bitcoin, il lui faudra attendre six blocs pour être certain que le payement ne sera pas invalidé par la suite (soit une heure en considérant dix minutes entre chaque bloc). La table 2.2 présente l'espérance qu'une double dépense aboutisse en fonction du nombre de blocs qui succédant la transaction et la puissance de l'attaquant. Comme nous le verrons dans la section suivante, posséder la moitié de la puissance de calcul du réseau coûte très cher financièrement. De plus, un dilemme du prisonnier [6] commence à se dessiner. En effet, si un mineur (ou un groupe de mineurs, appelé pool de mineurs) possède plus de la moitié de la puissance de calcul du réseau alors il enregistrera en moyenne plus de la moitié des blocs et donc il gagnera plus de la moitié des récompenses distribuées sur le réseau, ce qui peut rapidement devenir très rentable. Une attaque de double dépense provoquerait une diminution de confiance des utilisateurs envers le réseau entraînant une dévaluation de sa monnaie et donc une diminution de la valeur des récompenses touchées par le mineur. Généralement, lorsqu'un pool de mineurs possède une part trop importante de la puissance de calcul du réseau, il envoie certains de ses mineurs dans d'autres pools afin d'éviter une baisse de confiance sur le réseau et une dévaluation de sa monnaie.

2.5.2 Problèmes

Malgré sa sécurité élevée et sa décentralisation, Bitcoin présente de nombreux problèmes. Le premier d'entre eux est que le réseau bitcoin, avec le consensus qu'il propose, ne peut pas excéder 7 transactions par seconde. À titre de comparaison, Paypal effectue en moyenne 193 transactions par seconde et Visa environ 2315 transactions par seconde. De plus, les 7 transactions par seconde que propose Bitcoin ne sont en réalité jamais atteintes étant donné qu'un mineur n'est pas obligé de compléter un bloc pour l'enregistrer (un mineur peut très bien enregistrer un bloc sans autre transaction que celle qui le récompense). Bitcoin ne peut donc en aucun cas prétendre substituer le système bancaire mondial avec

6. https://en.wikipedia.org/wiki/Prisoner%27s_dilemma

	n									
q	1	2	3	4	5	6	7	8	9	10
2%	4%	0.237%	0.016%	0.001%	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$
4%	8%	0.934%	0.120%	0.016%	0.002%	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$
6%	12%	2.074%	0.394%	0.078%	0.016%	0.003%	0.001%	$\simeq 0$	$\simeq 0$	$\simeq 0$
8%	16%	3.635%	0.905%	0.235%	0.063%	0.017%	0.005%	0.001%	$\simeq 0$	$\simeq 0$
10%	20%	5.600%	1.712%	0.546%	0.178%	0.059%	0.020%	0.007%	0.002%	0.001%
12%	24%	7.949%	2.864%	1.074%	0.412%	0.161%	0.063%	0.025%	0.010%	0.004%
14%	28%	10.662%	4.400%	1.887%	0.828%	0.369%	0.166%	0.075%	0.034%	0.016%
16%	32%	13.722%	6.352%	3.050%	1.497%	0.745%	0.375%	0.190%	0.097%	0.050%
18%	36%	17.107%	8.741%	4.626%	2.499%	1.369%	0.758%	0.423%	0.237%	0.134%
20%	40%	20.800%	11.584%	6.669%	3.916%	2.331%	1.401%	0.848%	0.516%	0.316%
22%	44%	24.781%	14.887%	9.227%	5.828%	3.729%	2.407%	1.565%	1.023%	0.672%
24%	48%	29.030%	18.650%	12.339%	8.310%	5.664%	3.895%	2.696%	1.876%	1.311%
26%	52%	33.530%	22.868%	16.031%	11.427%	8.238%	5.988%	4.380%	3.220%	2.377%
28%	56%	38.259%	27.530%	20.319%	15.232%	11.539%	8.810%	6.766%	5.221%	4.044%
30%	60%	43.200%	32.616%	25.207%	19.762%	15.645%	12.475%	10.003%	8.055%	6.511%
32%	64%	48.333%	38.105%	30.687%	25.037%	20.611%	17.080%	14.226%	11.897%	9.983%
34%	68%	53.638%	43.970%	36.738%	31.058%	26.470%	22.695%	19.548%	16.900%	14.655%
36%	72%	59.098%	50.179%	43.330%	37.807%	33.226%	29.356%	26.044%	23.182%	20.692%
38%	76%	64.691%	56.698%	50.421%	45.245%	40.854%	37.062%	33.743%	30.811%	28.201%
40%	80%	70.400%	63.488%	57.958%	53.314%	49.300%	45.769%	42.621%	39.787%	37.218%
42%	84%	76.205%	70.508%	65.882%	61.938%	58.480%	55.390%	52.595%	50.042%	47.692%
44%	88%	82.086%	77.715%	74.125%	71.028%	68.282%	65.801%	63.530%	61.431%	59.478%
46%	92%	88.026%	85.064%	82.612%	80.480%	78.573%	76.836%	75.234%	73.742%	72.342%
48%	96%	94.003%	92.508%	91.264%	90.177%	89.201%	88.307%	87.478%	86.703%	85.972%
50%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

TABLE 2.2 – *Probabilité qu'une attaque de double dépense aboutisse en fonction de la proportion de puissance l'attaquant q et le nombre de blocs n qui succèdent au bloc d'où l'attaquant initie le fork. Source : Rosenfeld [65].*

un tel consensus.

Bitcoin présente un second problème majeur : sa consommation en énergie. En effet, la preuve de travail met en compétition les mineurs, les incitant ainsi à augmenter leur puissance de calcul pour augmenter leurs chances de gagner. Ainsi, au 31 Janvier 2021, le nombre de hashs journaliers effectués par les mineurs est estimé à 150 millions de terra hash (Th) [7], soit $1,5 \times 10^{20}$ hashs par jour ($\simeq 1,7 \times 10^{17}$ hashs par seconde). Actuellement, la consommation électrique totale du réseau Bitcoin est estimée à $111,48$ TWh par an[60]. Si Bitcoin était un pays, il serait le $32^{\text{ème}}$ plus grand consommateur d'électricité, devant la Nouvelle-Zélande ($108,80$ TWh par ans) et juste derrière les Émirats arabes unis ($113,20$ TWh par ans), une consommation proche du quart de la consommation électrique de la France ($450,80$ TWh par ans) [10]. De plus, étant donné que seul le premier mineur qui trouve un bloc valide peut l'enregistrer à la chaîne, l'énergie consommée par les autres mineurs sera perdue. La quantité d'énergie utilisée pour enregistrer une transaction est estimée à 648 kWh, soit la consommation moyenne d'un foyer américain en 23,5 jours. À titre de comparaison, Visa consomme en moyenne 149 kWh pour 100 000 transactions [22], une transaction bitcoin consomme donc en moyenne 460 000 fois plus d'énergie qu'une

transaction Visa (voir figure 2.9).

Le troisième point sur lequel le consensus de Bitcoin est souvent attaqué est qu'il incite à la centralisation. En effet, certains mineurs s'allient et combinent leur puissance de calcul pour augmenter leurs chances d'enregistrer un bloc et donc de gagner la récompense. Ces groupes de mineurs sont appelés pool de mineurs. Lorsqu'un pool de mineurs ajoute un bloc à la chaîne, la récompense est partagée entre les mineurs du pool au prorata de la puissance qu'ils fournissent, permettant ainsi de rendre les revenus des mineurs plus prévisibles. La décentralisation espéré par Satoshi Nakamoto est donc réduite par la création de pools de mineurs (une dizaine de pools de mineurs se partagent deux tiers de la puissance du réseau, voir figure 2.10). Toutefois, les mineurs ne peuvent pas créer des pools trop importants sinon le réseau deviendrait vulnérable à une attaque de double dépense entraînant une diminution de la confiance et donc de la valeur du Bitcoin, ce qui n'est évidement pas souhaitable pour les mineurs.

La dernière remarque que nous pouvons faire porte sur le fait que le consensus proposé par Bitcoin à tendance a inciter ses utilisateurs à investir dans du matériel afin d'augmenter leurs puissances de calcul (et donc leurs revenus). À long terme Bitcoin a tendance à exclure les utilisateurs ayant peu de puissance et de dissuader les nouveaux arrivants potentiels.

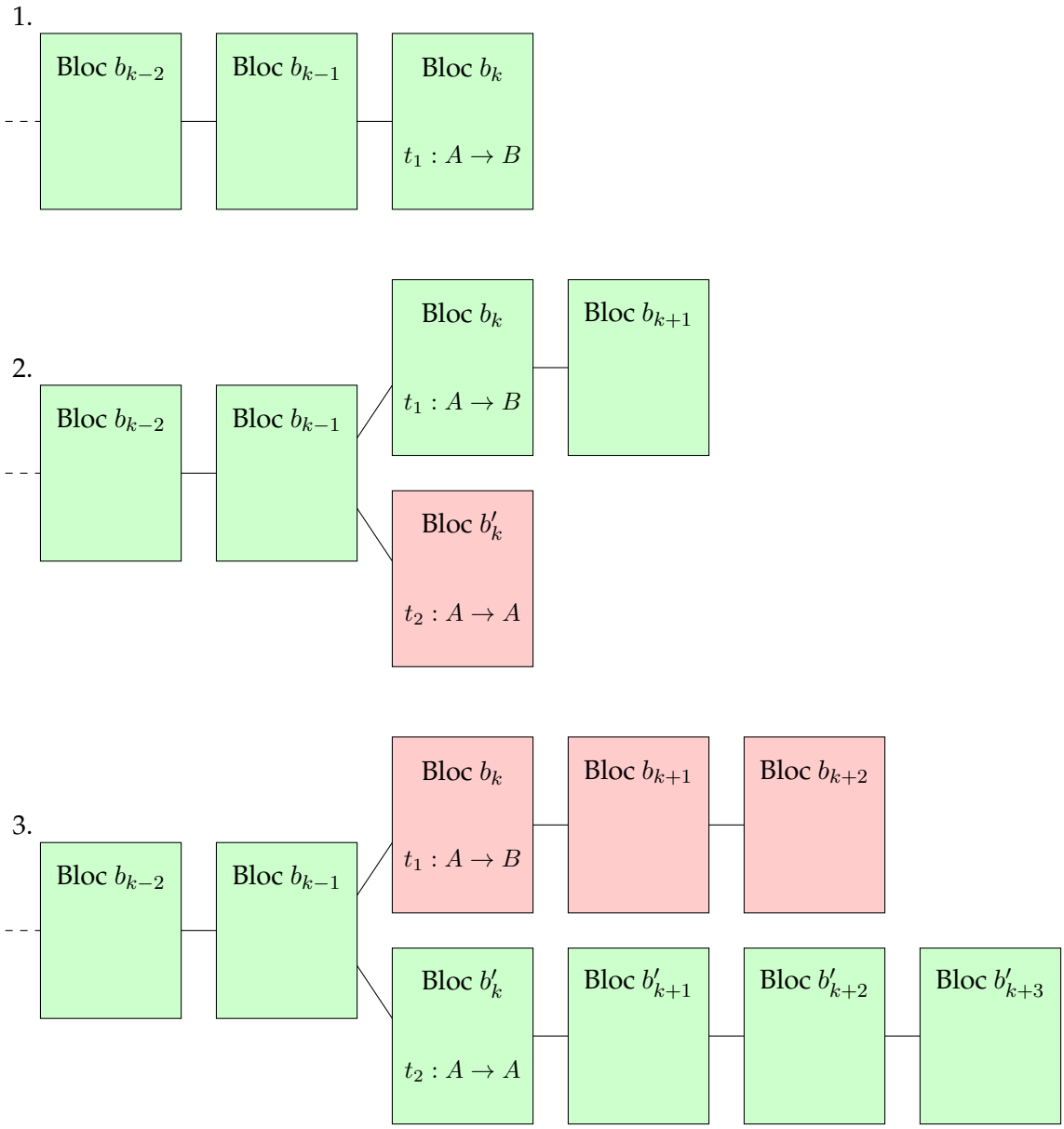


FIGURE 2.8 – *Déroulement de l'attaque de double dépense réalisée par un utilisateur A sur le réseau Bitcoin. Cette figure illustre les étapes décrites sous-section 2.5.1. Les blocs verts sont les blocs considérés par les nœuds complets, les rouges ceux qui ne le sont pas. Les étapes sont les suivantes : 1. l'attaquant A enregistre la transaction t_1 sur la chaîne. 2. A crée un fork au niveau du bloc b_{k-1} (le bloc précédent celui qui contient t_1) puis commence à calculer les blocs suivants (en incluant la transaction t_2), pendant ce temps les autres utilisateurs continuent d'enregistrer des blocs. 3. le fork de l'attaquant est maintenant plus grand que la chaîne actuelle, il l'envoie aux nœuds complets qui vont considérer le fork comme la chaîne principale, rendant orphelins les blocs de l'autre chaîne.*

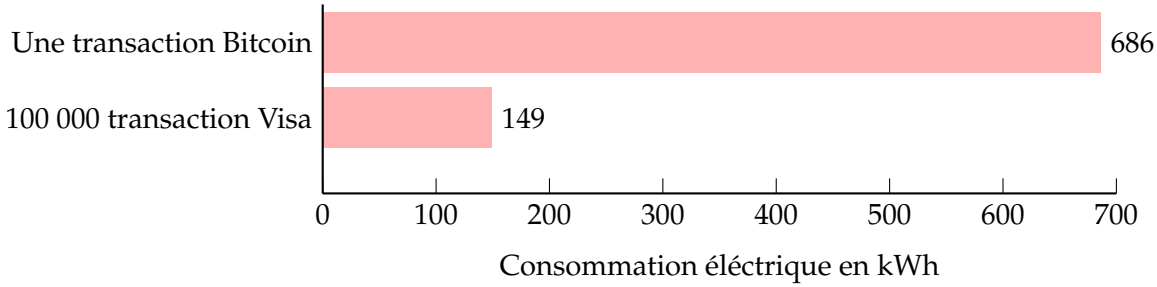


FIGURE 2.9 – *Comparaison de la consommation électrique moyenne entre une transaction Bitcoin et 100 000 transactions Visa [1].*

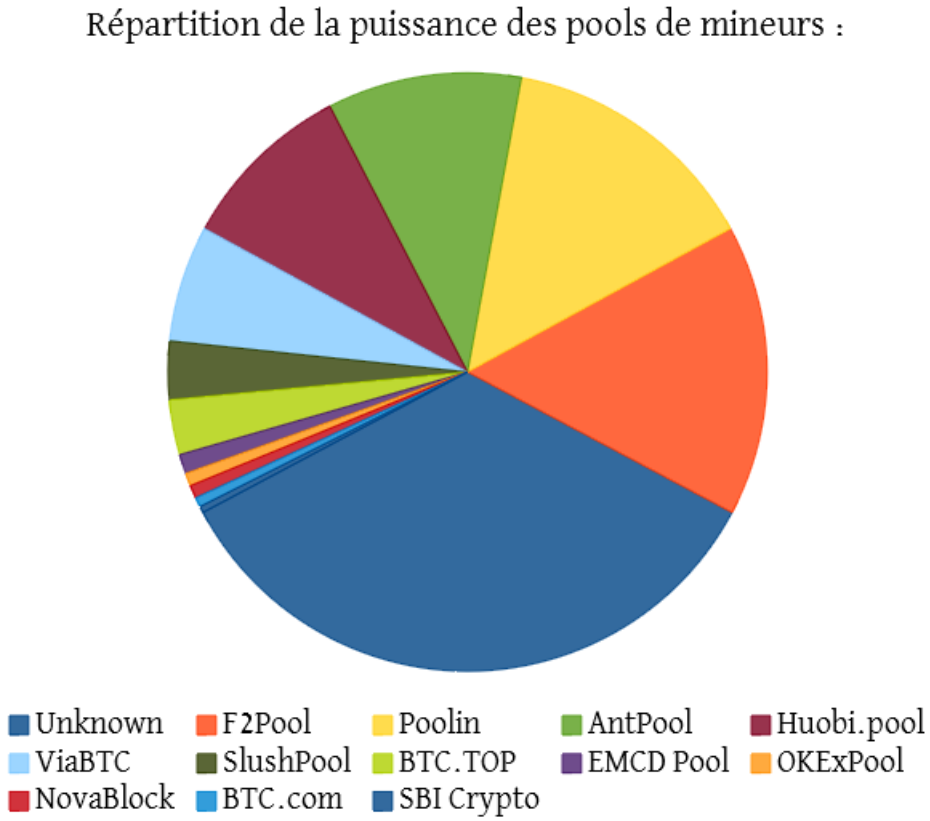


FIGURE 2.10 – *Répartition de la puissance des pools de mineurs du réseau Bitcoin le 01/02/2021 [6].*

III

Preuve d'utilisation

Dans ce chapitre, nous introduisons le protocole de preuve d'utilisation (Proof of Usage, PoU en anglais), un algorithme de consensus pensé pour une blockchain dont le but est de permettre l'échange de données privées entre entreprises (banque, assurance, etc) sous le contrôle de leur propriétaire. L'enregistrement de données sensibles n'étant pas possible sur une blockchain publique (voir Règlement Général sur la Protection des Données, RGPD [62]), le consensus de preuve d'utilisation s'inscrit dans une blockchain semi-privée. D'autres consensus comme la preuve de travail ou la preuve d'enjeu n'encourage pas ses utilisateurs à l'échange et à l'usage de la chaîne (la preuve d'enjeu pouvant même inciter l'inverse) alors que la valeur d'une monnaie dépend directement de son utilisation. La preuve d'utilisation propose donc d'inciter ses utilisateurs à l'utilisation de la chaîne et donc à réaliser des transactions ou à autoriser le partage de leurs données personnelles.

Le protocole de preuve d'utilisation a été pensé et développé par l'entreprise Pikcio pour sa blockchain PikcioChain [15, 47]. Notre travail à été d'analyser de décrire et de publier ce consensus.

3.1 Introduction

Comme nous avons vu dans le chapitre précédent, Bitcoin [58] est le premier système décentralisé proposant l'échange de valeurs (en l'occurrence de la crypto-monnaie).

Le consensus du Bitcoin nécessitant une consommation d'énergie colossale, Sunny King et Scott Nadal proposent la preuve d'enjeu en 2012 [44] (voir sous-section 1.3.5) afin de proposer un consensus moins énergivore. Plutôt que d'utiliser la puissance de calcul des mineurs pour ajouter des blocs à la chaîne actuelle et sécuriser le réseau, King et Nadal ont défini une méthode alternative dans laquelle un algorithme déterministe choisit un participant du réseau (i.e. un nœud) pour ajouter le bloc suivant. L'algorithme est basé

sur la sélection d'une pièce du réseau, le propriétaire de cette pièce sera le nœud qui aura le droit d'ajouter un bloc. Par exemple, s'il y a cent pièces sur le réseau, un nœud détenant dix pièces aura une chance sur dix d'être sélectionné pour ajouter le bloc suivant et gagner la récompense associée. Les nœuds détenant dix pièces ont dix fois plus de chances d'ajouter un bloc que les nœuds qui en possèdent une seule. Ainsi, plus un nœud stocke de pièces, plus la probabilité qu'il a d'ajouter un bloc à la chaîne est élevée. Il en résulte un consensus dans lequel les nœuds sont encouragés à économiser leur monnaie plutôt qu'à l'échanger.

Depuis de nombreux autres réseaux ont vu le jour, chacun proposant son propre consensus, incitant leurs utilisateurs à différents comportements. Par exemple, la preuve d'activité (Proof of Activity, PoA en anglais) proposé par Bentov et al. [5] en 2014 est un consensus mélangeant preuve de travail et preuve d'enjeu. Dans le consensus de preuve d'activité, les utilisateurs (i.e. les mineurs) tentent de calculer une preuve de travail pour ajouter le bloc suivant, le premier à trouver une preuve valide peut créer et enregistrer le bloc suivant et remporte la récompense associée. Toutefois, contrairement au consensus proposé par Bitcoin, le mineur qui enregistre le bloc ne remporte pas la totalité de la récompense. Un nombre fixe n de nœuds sont sélectionnés suivant la preuve d'enjeu de King et Nadal [44]. Ces utilisateurs doivent alors signer le bloc dans une fenêtre de temps limité les uns après les autres afin de prouver leur activité sur le réseau. Les frais de transactions du bloc sont partagés entre le nœud qui crée le bloc et les nœuds qui l'ont signé. La preuve d'activité incite donc ses utilisateurs à économiser (notamment à cause de la preuve d'enjeu) mais également à être tout le temps disponible (i.e. connecté) afin de signer les blocs et gagner les récompenses associées.

Nous pensons qu'il est important de privilégier l'échange sur le réseau plutôt que d'inciter ses utilisateurs à économiser. Ce critère nous paraît fondamental pour la durabilité de notre réseau et inciter l'échange de données plutôt que sa rétention. D'autres consensus proposent de privilégier l'échange, c'est le cas notamment de la preuve d'importance (Proof of Importance, PoI en anglais), algorithme introduit par NEM [14] et basé sur la preuve d'enjeu pondérée par un système de notations. Pour enregistrer le bloc suivant, un nœud doit avoir en sa possession un minimum de 10 000 pièces. De plus, chaque nœud a un score qui augmente avec le nombre de pièces qu'il possède et avec le nombre de transactions qu'il a effectué au cours des trente derniers jours. Plus les transactions qu'il a réalisées sont importantes et fréquentes, plus son score augmente et plus il aura de chance d'être sélectionné pour enregistrer le bloc suivant et gagner la récompense. Lorsqu'un nœud enregistre un bloc son score est remis à 0. La preuve d'importance est un algorithme de consensus qui incite ses utilisateurs à échanger plutôt qu'a économiser sa monnaie. Toutefois, la preuve d'importance est pensée pour une blockchain publique dans laquelle tout ses utilisateurs peuvent accéder à la chaîne et aux informations qu'elle contient.

Dans les blockchains publiques, généralement, les utilisateurs sont pseudonymisés et ne garantissent pas nécessairement l'anonymat. En effet, l'historique des transactions est décentralisé, accessible à tous. Les blockchains publiques ne peuvent donc pas garantir la confidentialité des opérations du réseau [66, 54]. La technologie blockchain ne permettant pas de supprimer des données, le droit à l'oubli pour ses utilisateurs ne peut pas

être respecté et donc l'enregistrement de données personnelles n'est pas autorisé sur une blockchain publique même si les données sont cryptées (en accord avec la réglementation européenne : Règlement Général sur la Protection des Données, RGPD [62]). Nous utiliserons une blockchain de type semi-privée, qui autorisent l'enregistrement de données personnelles.

En résumé, nous proposons un système d'échange de données personnelles qui incite ses utilisateurs à les partager. Notre système utilise une blockchain semi-privée dont seul un nombre restreint d'utilisateurs ont l'autorisation de consulter la chaîne ou y enregistrer de nouveaux blocs. Dans ce but, nous définissons un nouvel algorithme de consensus, la preuve d'utilisation, qui incite les utilisateurs du réseau à partager leurs données et à réaliser des transactions afin de garantir la stabilité du réseau. Nous commencerons par présenter les motivations qui nous ont poussé à proposer et à développer un tel système, avant d'en présenter son protocole. Enfin nous verrons comment les frais de transaction peuvent influencer le comportement des utilisateurs du système avant de voir les avantages et les inconvénients de celui-ci.

3.2 Motivations

Dans cette section nous présentons les motivations qui nous ont poussé à implémenter le protocole de preuve d'utilisation.

Actuellement, la quasi-totalité des prestataires de services reposent sur l'identification et la gestion de données personnelles, spécialement chez les prestataires tournés vers le numérique. Prenons l'exemple d'un client désirant réaliser un emprunt auprès d'une banque (figure 3.1). Pour commencer, le client va voir sa banque et lui fournit bon nombre de données personnelles (pièce d'identité, nom, âge, adresse, etc.). La banque va alors, vérifier, croiser et finalement valider les données du client afin d'être certaine de l'identité de son client. Ensuite, la banque va demander à ce même client de souscrire à une assurance afin d'être couvert en cas de non remboursement. Dans la plupart des cas, il n'y a pas de communication directe entre la banque et l'assureur. Le client va donc devoir redonner ses données à l'assureur, qui va les re-vérifier. Ainsi, on arrive dans un modèle où le client doit réaliser deux fois la même procédure et dans laquelle les données sont vérifiées deux fois (par la banque et par l'assurance). Ce modèle conduit à payer deux fois la certification des mêmes données et à une mauvaise expérience utilisateur.

Le protocole de preuve d'utilisation propose de simplifier ce modèle en établissant un lien de communication entre les banques et les assurances afin de leur permettre d'échanger les données certifiées de leurs clients contre rémunération. Si l'on reprend notre exemple, la banque pourrait alors vendre les données certifiées du client à l'assureur, générant ainsi un retour sur l'investissement de la vérification des données. De plus, en supposant que la banque vende les données certifiées moins chères que le prix de la certification, l'assureur serait également gagnant.

Lorsque l'on parle d'échanges de données, nous devons nous intéresser au modèle

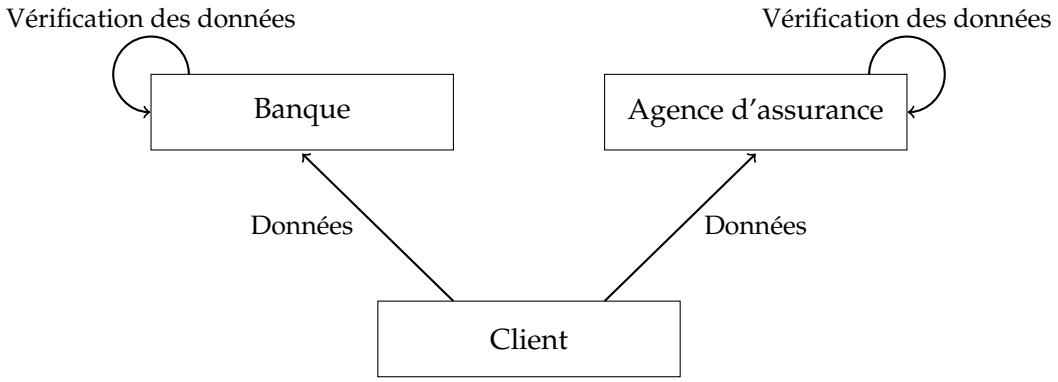


FIGURE 3.1 – *Modèle d'échanges de données entre un client, une banque et une agence d'assurance lorsque le client réalise un prêt.*

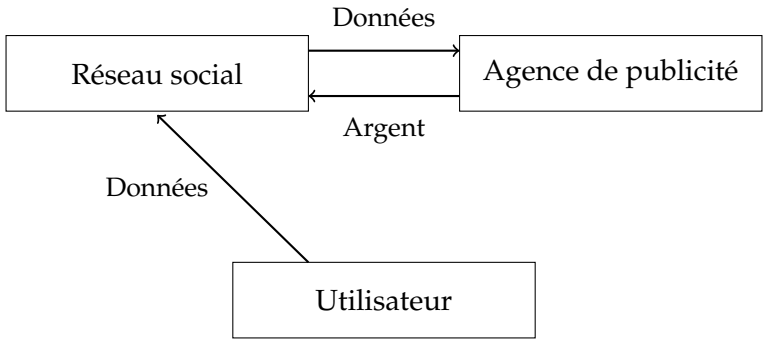


FIGURE 3.2 – *Modèle d'échanges de données entre un réseau social, une agence de publicité et un utilisateur du réseau social.*

utilisé par les agences de publicité (figure 3.2). Actuellement, les agences de publicité achètent des quantités importantes de données personnelles afin de définir ses cibles potentielles. Généralement, les données sont vendues par un tiers (réseau social, boîte mail, etc.), le propriétaire des données (l'utilisateur) est totalement exclu de la transaction et ne peut même pas contrôler le partage de ses propres données. Puisque le consentement de l'utilisateur n'est pas requis (ou est noyé dans des conditions d'utilisation trop longues et compliquées), le modèle n'inclue pas les utilisateurs, alors que ce sont eux les fournisseurs de la valeur. Seuls les vendeurs de données ou les agences de publicité gagnent de l'argent à partir des données des utilisateurs. Pour développer un modèle plus centré sur l'utilisateur, donnant à ce dernier le contrôle de ses données, deux directives peuvent être proposées :

1. Le vendeur des données (i.e. le réseau social) redistribue une part des revenus de la vente des données à ses utilisateurs.

2. Les coûts des données payés par les acheteurs de données (i.e. les agences de publicité) sont augmentés afin de reverser une partie à leurs propriétaires.

De tels modèles peuvent être qualifiés d'utopiques, étant donné qu'il y a peu de chance que le vendeur soit prêt à diminuer ses revenus et/ou que l'acheteur soit d'accord pour augmenter ses dépenses.

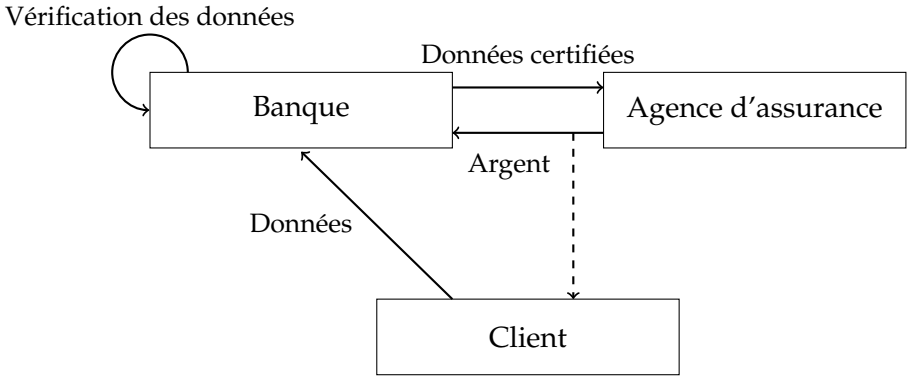


FIGURE 3.3 – *Modèle d'échanges de données entre un client, une banque et une agence d'assurance lorsque le client réalise un prêt avec un système basé sur la preuve d'utilisation. "Données certifiées" sont les données personnelles du client vérifiées par la banque.*

Le modèle que propose le protocole de preuve d'utilisation (figure 3.3) est un modèle d'échange de données certifiées entre plusieurs entités (banque, assurance, etc.) centré sur l'utilisateur. Ce modèle propose à l'utilisateur de partager ou non ses données, lui permettant ainsi de reprendre le contrôle celles-ci. Notre modèle est une alternative au modèle bancaire (figure 3.1), basé sur le modèle des agences de publicité (figure 3.2) auquel on ajoute un droit de contrôle sur leurs données aux utilisateurs. Dans notre modèle, un utilisateur voulant réaliser un emprunt doit encore fournir ses données personnelles à la banque et celle-ci doit encore vérifier les données renseignées. Toutefois, lorsque cet utilisateur demande une assurance, l'assureur peut proposer à la banque d'acheter les données certifiées en échange d'un paiement (avec l'accord de l'utilisateur). Une partie de la transaction revenant à l'utilisateur. Dans un tel modèle, les entreprises (i.e. les banques et les agences d'assurance) peuvent accéder aux données certifiées (après l'achat), ce qui implique moins de frais de vérification des données, ainsi qu'un gain de temps pour les entreprises et l'utilisateur. Dans ce modèle, l'entreprise qui a traité et certifié les données de l'utilisateur bénéficie d'un retour sur investissement pour son travail de validation. De plus, l'inclusion de l'utilisateur dans le processus de validation des transactions de ses données personnelles est un gage de confiance dans la relation entre le client et l'entreprise. Un utilisateur est incité à donner son accord par le biais d'une possible rémunération.

Étant donné que les données enregistrées sur la blockchain sont des données privées et sensibles, ce modèle ne peut pas être envisagé pour une blockchain publique (voir le règlement général sur la protection des données [62] pour plus de détails). Nous présentons donc un protocole de preuve d'utilisation pour une blockchain semi-privée.

3.3 Protocole

Dans cette section nous détaillons le fonctionnement de la preuve d'utilisation, qui peut être considéré comme une extension du travail de Bentov et al. [5] sur la preuve d'activité. La principale différence entre ces deux protocoles réside dans l'incitation qu'ils donnent à leurs utilisateurs. La preuve d'utilisation incite les utilisateurs à échanger leur

monnaie alors que la preuve d'activité incite ses utilisateurs à être connectés et à stocker leur monnaie. Le protocole de la preuve d'utilisation propose de récompenser un utilisateur aléatoire parmi ceux qui sont à l'origine d'une ou plusieurs transactions enregistrées dans le dernier bloc enregistré.

Dans notre système, nous appelons administrateur l'utilisateur qui est en charge de créer, d'initialiser, de contrôler et d'organiser le réseau. Pour accéder au réseau, un utilisateur doit obtenir l'accord de l'administrateur. En effet, lorsqu'un utilisateur veut accéder au réseau pour la première fois, il doit en faire la demande à l'administrateur et lui fournir une identité numérique complète (comprenant un document d'identité officiel). L'administrateur peut choisir d'accepter ou non un utilisateur suivant un processus de Know-Your-Costumer (KYC) [29] qui garantit l'identité de l'utilisateur, réduisant considérablement le risque d'attaques externes sur le système, mais ne le supprimant pas complètement. Sur le réseau proposé nous supposerons l'existence d'une bijection entre les identités et les entités. Autrement dit, un utilisateur réel ne peut pas créer plusieurs comptes sur le réseau. Le processus de KYC permet également à notre système d'être en accord avec le lois internationales de lutte contre le blanchiment d'argent [57].

Comme dit précédemment, la preuve d'utilisation est prévue pour une blockchain privée dans laquelle seulement certains sites prédéfinis sont autorisés à consulter et écrire sur la chaîne. Nous appelons ces sites des nœuds complets. Les nœuds complets sont chargés de vérifier la validité des transactions et sont récompensés pour ajouter des blocs de transaction à la chaîne. Les nœuds complets sont les seuls nœuds à partager la blockchain. N'importe quel nœud peut devenir nœud complet en envoyant une requête à l'administrateur du réseau en échange d'une caution qu'il récupérera quand il ne sera plus nœud complet à condition qu'il n'ait pas été malveillant en tant que nœud complet. Toutefois, le réseau disposant d'un nombre fixe k de nœuds complets, tous les nœuds ne peuvent pas devenir nœud complet en même temps, seulement les k nœuds proposant la caution la plus importante deviennent nœud complet (comme une enchère). Si un nœud complet n'agit pas correctement pour le réseau (i.e. il n'enregistre plus de bloc, il essaie d'attaquer ou de corrompre le réseau) l'administrateur le remplace par un autre nœud du réseau et sa caution sera perdue. Si un nœud complet veut se retirer, il doit en faire la demande à l'administrateur pour qu'il soit remplacé et que sa caution lui soit rendue. Autrement, un nœud complet reste nœud complet aussi longtemps qu'il agit dans les règles et l'intérêt du réseau. Le nombre de nœuds complets k est défini par l'administrateur et va avoir une influence sur le niveau de décentralisation et la vitesse du réseau. Plus k est grand, plus le réseau est décentralisé (i.e. plus il est tolérant aux pannes et résistant aux attaques) plus il est lent. À l'inverse, plus k est petit, plus le réseau sera rapide et moins le réseau sera sécurisé.

Pour initialiser le réseau, l'administrateur génère k adresses pour les nœuds complets, chacune d'elle correspondant à un nombre binaire de taille 256.

Dans le but de présenter le plus clairement possible le protocole de la preuve d'utilisation, nous proposons de décrire une fonction de changement d'états $changeState(S, T) = S'$ qui prend en entrée l'état courant de la blockchain S, un ensemble de transactions $T = \{t_0, \ldots, t_{|T|-1}\}$ et renvoie un nouvel état S' correspondant à S après l'ajout des transactions de T. Autrement dit, à chaque itération de la fonction $changeState$ un bloc est

créé et enregistré sur la chaîne (s'il est valide).

Considérons $M = \{m_0, \ldots, m_{k-1}\}$ l'ensemble des k nœuds complets et U l'ensemble de nœuds. La fonction de changement d'état du protocole de la preuve d'utilisation est définie comme suit :

1. **Sélection du nœud complet :** sélection du nœud complet $m_s \in M$ chargé de créer, d'ajouter et de partager le bloc suivant.
2. **Sélection des transactions :** m_s sélectionne des transactions parmi la liste de transactions en cours afin de créer un ensemble de transactions valides T_B.
3. **Sélection du nœud gagnant :** m_s sélectionne le "nœud gagnant" - le nœud (i.e. l'utilisateur) qui va gagner la récompense du bloc.
4. **Partage du bloc :** m_s crée le bloc contenant les transactions T_B, l'ajoute à sa chaîne et le partage à travers le réseau.

La fonction $changeState$ est une fonction cyclique, elle est exécutée à intervalle de temps régulier. Dans la suite de cette partie nous utilisons la fonction de hachage SHA-256 qui renvoie un hash de 256 bits.

Les sections suivantes détaillent les étapes du protocole précédemment cités.

3.3.1 Sélection du nœud complet

La première étape avant d'ajouter un bloc à la chaîne dans le protocole de la preuve d'utilisation est de sélectionner un nœud complet $m_s \in M$ qui sera chargé de le créer, le signer puis le partager sur le réseau. La sélection de m_s est réalisée en comparant le hash du bloc précédent avec l'ensemble des IDs des nœuds complets (voir figure 3.4). Le nœud complet ayant l'identifiant le plus proche du hash du bloc précédant est choisi pour être le prochain nœud à enregistrer un bloc à la chaîne. Étant donné que tous les nœuds complets connaissent le hash du bloc précédent ainsi que tous les IDs des nœuds complets, ils sont capables de se mettre d'accord sur m_s sans aucune communication.

On peut remarquer (notamment dans la figure 3.4) que tout les nœuds complets n'ont pas une probabilité équivalente d'être sélectionnés (même la fonction de hachage retourne des hashs uniformément répartis). Par exemple, dans la figure 3.4, m_1 a moins de chance d'être sélectionné que m_0 ou m_2. Ce manque d'équité est contrecarré par le fait que les nœuds complets sont distribués via un système d'enchères et que les identifiants ayant la plus grande probabilité d'être sélectionnés sont distribués aux nœuds qui ont proposé la plus grosse caution.

Comme mentionné précédemment, la fonction $changeState$ est une fonction cyclique qui est exécutée à intervalle de temps régulier. Si le nœud complet m_s sélectionné ne partage pas de bloc après un nombre fixe d'exécutions, la fonction le considérera comme

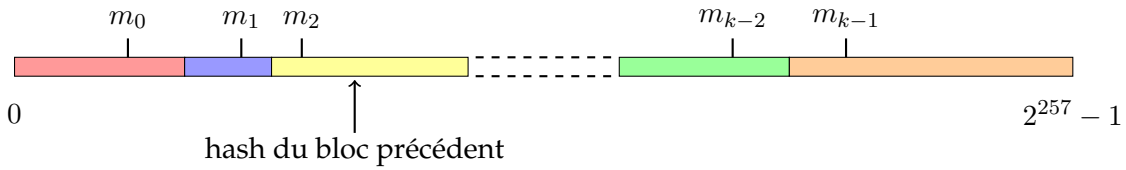


FIGURE 3.4 – *Exemple de sélection de m_s. Les bornes 0 et $2^{257} - 1$ correspondent respectivement à la valeur minimale et maximale de la sortie de la fonction de hachage SHA-256, la droite représente l'ensemble des valeurs possibles de retour de cette fonction. Chaque nœud maître est placé sur la valeur correspondante à son ID. Les zones colorées correspondent à l'ensemble des valeurs pour lequel un nœud maître est choisi. Par exemple la zone bleu correspond à l'ensemble des valeurs du bloc précédent pour lesquelles m_1 est choisi. Dans l'exemple représenté sur la figure, la hash du bloc précédent et la répartition des identifiants font que m_2 sera choisi pour ajouter le prochain bloc.*

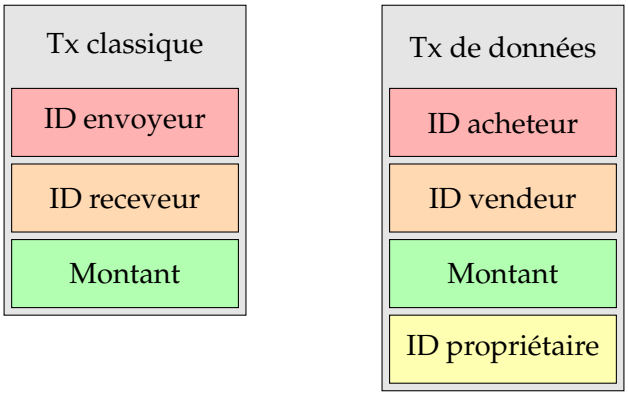


FIGURE 3.5 – *Modèles des transactions utilisées par le protocole de Proof of Usage. Une transaction classique se compose de l'identifiant de l'envoyeur et du receveur ainsi que le montant de la transaction. Une transaction de données se compose de l'identifiant de l'acheteur des données, de l'identifiant de leur vendeur, le montant auxquelles elles sont vendu et l'identifiant du propriétaire des données.*

indisponible. À l'exécution suivante, la fonction ignorera son identifiant et sélectionnera ainsi le second identifiant le plus proche du hash du bloc précédent. Cette règle prémunit le réseau de tomber dans un état bloquant où m_s n'est pas disponible pour une raison ou une autre (déconnexion, latence sur le réseau, etc...).

3.3.2 Sélection des transactions

Nous proposons deux types de transactions pour la preuve d'utilisation : les transactions dites "classiques" et les transactions de données. La structure de ces deux types de transactions est présentée figure 3.5.

- Une transaction classique permet à un nœud A d'envoyer une somme (i.e. un nombre de pièces du réseau) à un autre nœud B.

- Une transaction de données permet à un nœud A d'acheter les données certifiées d'un nœud C à un nœud B en échange de monnaie du réseau. Par exemple, une

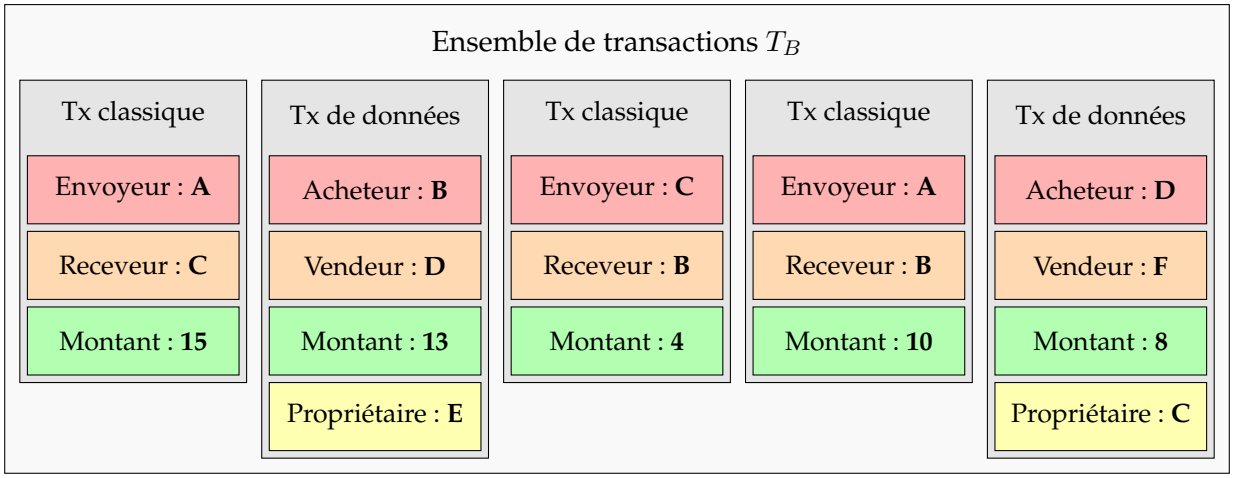


FIGURE 3.6 – *Exemple d'ensemble de transactions T_B d'un bloc.*

banque A peut acheter les données d'un client C à une assurance B qui a déjà vérifié la véracité des données. Une transaction de données ne peut pas être initialisée sans l'accord de leur propriétaire (i.e. A et B ne peuvent pas réaliser la transaction tant que C ne l'a pas signée avec sa clé privée).

Quand un nœud initialise une transaction, il l'envoie à tous les nœuds complets pour qu'elle soit enregistrée sur la chaîne. Quand une transaction de données est initialisée et acceptée par le propriétaire des données, la transaction reste dans un état "attente de réception des données" avant d'être enregistrée sur la chaîne jusqu'à ce que l'acheteur reçoive les données. L'état "attente de réception des données" permet de prémunir le réseau de nœuds malveillants qui essayeraient d'être payés sans posséder les données (ou sans les envoyer).

Quand un nœud complet reçoit une transaction, il vérifie sa validité (les signatures sont correctes et le nœud possède un solde supérieur à la somme envoyée) puis l'enregistre localement sur un liste de transactions en attente d'être enregistrées. Ainsi, chaque nœud complet possède une liste locale de transactions en attente dans laquelle il choisit les transactions à ajouter dans le bloc suivant quand il est sélectionné.

Quand la fonction $changeState$ sélectionne un nœud complet m_s pour créer le bloc suivant, m_s constitue un ensemble de transactions T_B à partir de sa liste locale de transactions en attente. Ces transactions constitueront le bloc suivant. Idéalement, pour limiter l'attente de validation globale des transactions, m_s devrait ajouter les transactions dans l'ordre où il les a reçues (comme dans une file). Malheureusement, du fait d'un système de communication asynchrone, les autres nœuds complets ne sont pas en capacité de vérifier l'ordre dans lequel m_s a reçu les transactions. Toutefois, ce phénomène ne représente pas un problème majeur étant donné que même si m_s omet une transaction (volontairement ou non), un autre nœud complet pourra l'inclure dans un futur bloc et la seule conséquence sera d'allonger le temps de validation de cette transaction.

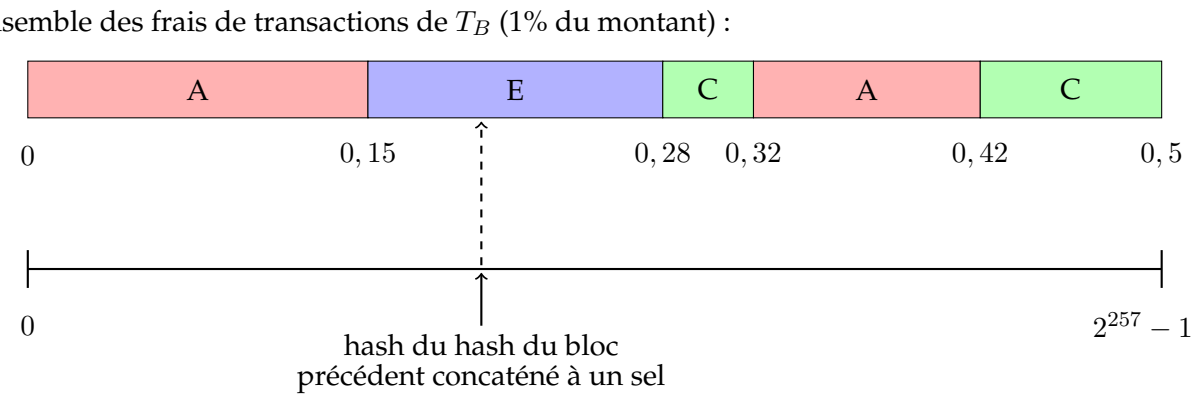


FIGURE 3.7 – *Exemple de sélection d'un nœud gagnant à partir du bloc de la figure 3.6 en considérant des frais de transactions de 1%. La première ligne correspond aux frais des transactions de l'ensemble T_B de la figure 3.6 que l'on ajoute consécutivement en renseignant l'initiateur de la transaction (ou le propriétaire des données dans le cas d'une transaction de données). Les bornes 0 et $2^{257} - 1$ correspondent respectivement à la valeur minimale et maximale de la sortie de la fonction de hachage SHA-256. Pour définir le nœud gagnant, le bloc précédent est concaténé à un sel, haché puis mis à l'échelle des frais de transactions (par une règle de trois). Le nœud gagnant est l'initiateur de la transaction (ou le propriétaire des données dans le cas d'une transaction de données) ayant généré les frais correspondants. Dans cet exemple, un seul nœud gagnant est sélectionné :* ***E****.*

3.3.3 Sélection des nœuds gagnants

Comme dans le protocole de la preuve d'activité, la preuve d'utilisation sélectionne des nœuds (i.e. des utilisateurs), appelé nœuds gagnants, qui seront récompensés en gagnant une partie des frais de transactions du dernier bloc enregistré. Dans la preuve d'activité, les nœuds sélectionnés doivent être en ligne quand le bloc est ajouté pour recevoir la récompense. Dans la preuve d'utilisation, un nœud gagnant doit être à l'origine d'une transaction (ou propriétaire des données échangées dans le cas d'une transaction de données). Afin de sélectionner les n nœuds gagnants qui partageront la récompense, le nœud complet m_s calcule n hashs à partir du hash du bloc précédent concaténé successivement à n sels cryptographiques. Les hashs obtenus sont mis à l'échelle des frais de transactions totaux du bloc puis comparés avec les frais de transaction (voir figure 3.7). Les n nœuds ayant initialisé ces transactions seront les nœuds gagnants (excepté pour les transactions de données, pour lesquelles ce sera le propriétaire des données qui sera récompensé). Notons qu'un nœud peut être sélectionné plusieurs fois (par le biais d'une ou plusieurs transactions) et donc gagner plusieurs parts de la récompense. Le hash du bloc précédent et les sels étant connus par l'ensemble des nœuds complets, les hashs obtenus peuvent être recalculés et vérifiés par tous les nœuds complets sans aucune communication.

Concernant le nombre n de nœuds à sélectionner, nous avons choisi de prendre une valeur fixe mais il est possible le définir en tant que variable grandissant avec le nombre de transactions du bloc et/ou la somme totale des frais de transaction du bloc par exemple.

3.3.4 Partage du bloc

À cette étape, m_s a construit l'ensemble des transactions T_B à inclure dans le prochain bloc B. Maintenant, m_s construit l'en-tête du bloc contenant : le hash du bloc précédent, la racine de l'arbre de Merkle construit à partir des transactions de T_B, la liste des identifiants des nœuds gagnants et l'horodatage du bloc. L'en-tête est ensuite signée par m_s avec sa clé privée afin de la bloquer définitivement et la sécuriser. Le hash d'un bloc B correspond au hash de son en-tête concaténé avec l'ensemble de transactions qu'il contient (T_B).

Si m_s a réalisé correctement les étapes précédentes, actuellement, il a correctement créé et ajouté le bloc B à sa chaîne (voir figure 3.8) et peut maintenant le partager avec l'ensemble des autres nœuds complets. Lorsqu'un nœud complet reçoit un bloc, il doit vérifier la validité de son en-tête et des transactions qu'il contient avant de l'ajouter à sa propre chaîne. Un nœud complet m_r recevant un bloc B le considère comme valide si et seulement si les conditions suivantes sont respectées :

1. le hash précédent inscrit dans l'en-tête du bloc B correspond au hash du dernier bloc enregistré dans la chaîne de m_r,
2. l'identifiant du nœud complet sélectionné m_s est l'identifiant le plus proche du hash du bloc précédent (i.e. m_s est bel et bien le nœud complet sensé enregistrer le bloc suivant),
3. toutes les transactions du bloc B (contenues dans T_B) sont valides (i.e. elles sont toutes signées et les envoyeurs / acheteurs possèdent un solde supérieur au montant de la transaction),
4. la sélection des nœuds gagnants n'est pas biaisée (i.e. m_s récompense les bon nœuds).

Après avoir reçu un bloc valide, m_r l'ajoute à sa chaîne et supprime de sa liste de transactions en attente les transactions ainsi validées. La condition 1. peut ne pas être validée pour les deux raisons suivantes : (i) m_s essaie de partager un bloc contenant de mauvaises informations et (ii) m_r n'est pas à jour et son dernier bloc est un bloc plus vieux dans la chaîne de m_s. Le cas (ii) peut apparaître si m_r n'a pas reçu le(s) précédent(s) bloc(s), par exemple s'il se reconnecte au réseau après avoir été inactif ou en panne. Si la condition 1. n'est pas respectée, m_r réalise un protocole de récupération pour tenter de récupérer les blocs manquants (si toutefois ils existent), c'est à dire que l'on soit dans le cas (ii). Soit h, le hash du dernier bloc B de la chaîne de m_r. Le protocole de récupération est exécuté comme suit :

1. m_r demande à tous les autres nœuds complets le hash du dernier bloc de leur chaîne.
2. Les nœuds complets envoient le hash de leur dernier bloc à m_r.
3. m_r sélectionne le hash h' qui lui est retourné en majorité et le compare avec le hash de son dernier bloc h. Si $h' = h$, alors m_r est à jour, le protocole de récupération est terminé (i.e. on est dans le cas (i)). Si $h' \neq h$, on se trouve dans le cas (ii), le protocole continue avec les étapes suivantes.

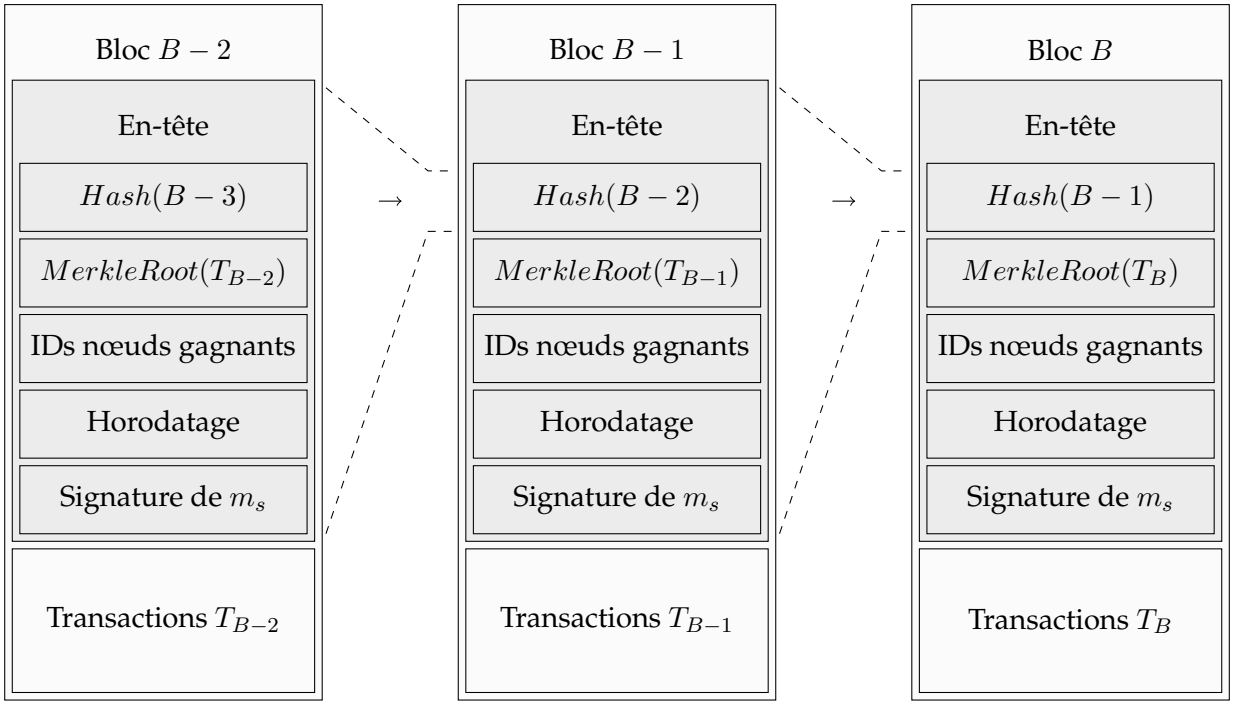


FIGURE 3.8 – *Structure de la chaîne du protocole de Proof of Usage.*

4. m_r demande l'ensemble des blocs manquants entre le bloc correspondant au hash reçu h' et le dernier bloc de sa chaîne à un nœud complet à jour (i.e. un nœud complet qui lui a envoyé h'). La numérotation des blocs permet à m_r de demander facilement les blocs manquants en envoyant une requête contenant le numéro de son dernier bloc.

5. m_r valide les blocs reçus un à un comme vu précédemment.

Le protocole de récupération permet à m_r de reconstruire la partie manquante de sa chaîne. Remarquons que la chaîne reste à jour tant que au moins la majorité relative des nœuds complets sont à jour. Cette propriété de sécurité est une conséquence directe du protocole de récupération : quand un nœud complet m_r n'est pas à jour, il demande le hash du dernier bloc et se met à jour en fonction du hash qui lui est renvoyé le plus grand nombre de fois. Donc si une majorité relative des nœuds complets lui renvoient le hash du dernier bloc de la chaîne à jour, alors m_r sera correctement à jour.

3.4 Frais de transaction

Le comportement des utilisateurs du réseau peut être influencé par le système de frais de transaction choisi. En effet, selon s'il y a des frais de transaction ou non, les utilisateurs peuvent avoir plus ou moins de libertés dans leurs échanges. Dans cette section nous présentons deux modèles : le premier avec des frais sur les transactions et le second sans frais. Nous présenterons les avantages et les inconvénients que ces systèmes représentent.

3.4.1 Modèle avec frais de transaction

La plupart des crypto-monnaies publiques utilisent un système basé sur des frais de transactions qui sont récupérés par le nœud qui ajoute le bloc contenant ces frais. Dans le premier modèle que nous proposons, à chaque transaction, un pourcentage de la somme totale est prélevé et sera redistribué lorsque la transaction sera enregistrée dans un bloc. Ainsi, chaque fois qu'un nœud maître ajoute un bloc, il gagne un pourcentage de la récompense totale et redistribue équitablement le reste aux nœuds gagnants. Un tel système fonctionne généralement avec une monnaie déflationniste, mais il peut être adapté à une monnaie inflationniste en générant de nouvelles pièces à chaque bloc en plus des frais de transactions.

Le principal problème d'un tel modèle est que les utilisateurs de systèmes bancaires classiques ne sont pas familiers avec les frais de transactions et les systèmes utilisés en crypto-monnaie. Dans un modèle bancaire classique, lorsqu'un utilisateur envoie 10 pièces à un autre, le bénéficiaire reçoit la totalité de la somme. L'utilisateur qui a envoyé les pièces ne s'attend pas à payer des frais supplémentaires pour le service.

3.4.2 Modèle sans frais de transaction

Un modèle sans frais de transaction implique forcément que la récompense distribuée pour l'enregistrement des blocs est générée à chaque bloc. Un tel système est inévitablement inflationniste mais lorsqu'un utilisateur A envoie 10 pièces à un autre B, A paie 10 pièces et B reçoit 10 pièces (comme dans un système bancaire classique).

Toutefois, l'absence de frais de transactions ne permet pas d'empêcher les nœuds d'inonder le réseau en initiant une grande quantité de transactions. Par exemple, si deux utilisateurs se font passer en boucle une grande quantité de pièces d'un compte à l'autre, ils peuvent prétendre à participer à la preuve d'utilisation et potentiellement gagner de nouvelles pièces sans jamais risquer de perdre les leurs. Par conséquent, le système se retrouverait à réaliser des transactions inutiles, ralentissant les véritables transactions et diminuant la récompense distribuée aux utilisateurs honnêtes.

Une des solutions à ce problème consiste à limiter les transactions de chaque nœud dans le temps, comme dans un système bancaire classique. Cette limite peut être définie en fonction du nombre de transactions et/ou du montant total autorisé des transactions pour une période donnée. Des frais de transaction sont appliqués aux transactions dépassant la limite définie. Dans ce cas là, les nœuds malveillants ne peuvent plus inonder le réseau de transactions car ils dépenseront rapidement des frais de transaction qu'ils risquent de ne jamais récupérer.

De notre point de vue, un système créant de nouvelles pièces sur le réseau lors de la création de nouveaux blocs et une absence de frais de transactions serait le meilleur système. Toutefois pour des raisons de sécurité que nous abordons dans la section suivante, il est nécessaire de limiter les utilisateurs par des frais de transactions. C'est pourquoi nous proposons un système qui propose aux utilisateurs de réaliser des transactions sans frais

sauf s'il dépasse une certaine quantité de pièces envoyées dans un temps donné (comme pour un système bancaire classique). Les transactions suivantes sont alors soumises à des frais de transactions. De plus nous proposons aux utilisateurs la possibilité de réaliser une transaction importante (sans limite et sans frais) tous les x temps. Si un utilisateur réalise une seconde transaction importante sans que le temps nécessaire entre les deux soit écoulé, des frais seront prélevés sur la seconde transaction.

3.5 Sécurité

Cette section décrit le comportement d'un réseau utilisant la preuve d'utilisation face aux attaques d'utilisateurs malveillants.

Les attaques en provenance des nœuds complets ne sont pas pris en compte car nous les supposons bienveillants. En effet, ils sont normalement dignes de confiance et le risque d'une attaque de leur part est réduit par plusieurs mécanismes : l'administrateur connaît l'identité réelle de ces nœuds (KYC) et ils mettent en caution une somme importante (qui leur sera retirée en cas de mauvais comportement). De plus, la preuve d'utilisation est pensée pour les blockchains semi-privées uniquement, les utilisateurs sont soumis à une authentification forte lors de leurs première connexion (un document d'identité officiel est requis pour le KYC) et ils ne peuvent donc pas se cacher derrière un pseudonyme. Ce qui nous amène à notre seconde supposition : une entité réelle ne peut pas créer plusieurs identités sur le réseau.

3.5.1 Attaque par déni de service

Une attaque par déni de service (DoS), ou attaque par inondation, consiste à rendre une machine ou une ressource réseau indisponible pour les nœuds auxquels elle est destinée en l'inondant de demandes superflues. Les attaques DoS distribuées (DDoS) sont une extension des attaques DoS dans lesquelles la ressource ciblée est inondée à partir de nombreuses sources différentes, rendant impossible de stopper l'attaque simplement en bloquant une unique source. Les attaques DoS et DDoS sont devenues l'un des principaux problèmes de sécurité sur Internet [70] et représentent un coût important pour le secteur de services en ligne.

Sur un réseau d'échanges de valeurs, les attaques DoS ou DDoS consistent principalement à inonder le réseau de transactions factices pour augmenter le temps de vérification des transactions afin de bloquer les véritables transactions. Le processus de KYC empêche toutefois un utilisateur de créer plusieurs identités (plusieurs nœuds) et prévient donc des attaque DDoS (à moins qu'un nombre conséquent d'utilisateurs ne choisissent de s'unir). Le processus de KYC permet également à l'administrateur de connaître l'identité réelle de tous ses utilisateurs et lui permet de bloquer toutes les transactions en provenance d'un nœud à l'origine d'une attaque DoS (les nœuds complets peuvent facilement repérer l'attaquant et le mettre sur liste noire). Dans ce cas, le nœud peut être exclu du réseau et

perdre ses pièces.

3.5.2 Problème de double dépense

Comme nous avons vu dans la sous-section 2.5.1, seuls les nœuds ou les groupes de nœuds possédant plus de 50% de la capacité d'écriture sur le réseau peuvent réaliser à coup sûr une attaque par double dépense sur une blockchain ne considérant que la chaîne la plus longue.

Avec le protocole de preuve d'utilisation, la puissance d'écriture est détenue par les nœuds complets. Ils sont les seuls à pouvoir lancer une attaque par double dépense. Dans cette section, nous supposons que les nœuds complets sont de confiance. Toutefois, que se passe-t-il si l'un d'entre eux (ou un petit nombre) est malveillant ? Les nœuds complets gagnent des l'argent en enregistrant des blocs valides à la chaîne et ils peuvent être bannis et remplacés s'ils ne le font pas au moment où c'est à eux de le faire. Dans ce cas là, le nœud complet perd des pièces en n'ajoutant pas de blocs et peut perdre sa caution à terme. De plus, les nœuds complets sont des utilisateurs possédant beaucoup de pièces sur le réseau (entre leur caution et leur rémunération) et ils n'ont pas intérêt à effectuer une attaque qui provoquerait certainement une dévaluation de la monnaie et donc une perte importante pour eux.

3.5.3 Pouvoir des pool d'utilisateurs

Les pools d'utilisateurs sont des groupes d'utilisateurs qui s'unissent pour augmenter leur puissance sur la blockchain. Le pouvoir d'un utilisateur sur la chaîne dépend du consensus utilisé sur le réseau. Par exemple, dans le cas de la preuve de travail (tel que Bitcoin), la puissance d'un nœud dépend directement de sa capacité de calcul. Pour la preuve d'enjeu basée sur la richesse de l'utilisateur, la puissance d'un nœud dépend du nombre de pièces qu'il possède. Pour la preuve d'utilisation, la puissance d'un nœud dépend de la quantité de pièces qu'il a utilisée récemment (dans le dernier bloc). En effet, pour augmenter ses chances d'être sélectionné en tant que nœud gagnant, le nœud doit augmenter la somme totale de pièces qu'il a envoyée (ou permis d'envoyer, en cas de transaction de données) dans le dernier bloc. Ainsi, si les utilisateurs d'un pool veulent leurs gains tout en ne payant pas de frais, ils doivent stocker l'ensemble des pièces des utilisateurs du pool sur un seul nœud et les envoyer d'un nœud à l'autre pour générer de grosses transactions.

3.5.4 Tolérance aux pannes

Tout nœud (dont les nœuds complets) peuvent être défectueux et peuvent se déconnecter ou souffrir de la latence du réseau. Avec le protocole de preuve d'utilisation, lorsqu'un nœud complet reçoit un bloc dont le hash du bloc précédent ne correspond pas au

hash de son dernier bloc (le nœud n'est pas à jour à cause d'une déconnexion ou autre), le nœud complet exécute le protocole de récupération (voir sous-section 3.3.4). Dans ce protocole, le nœud complet demande l'état de la chaîne à tous les autres nœuds complets et se met à jour en accord avec la majorité relative. Ainsi, le réseau reste fiable tant qu'une majorité relative des nœuds complets sont à jour. Quand un nœud complet est de retour sur le réseau après une déconnexion ou une panne, il peut se remettre correctement à jour tant que la majorité relative des nœuds complets le sont.

3.6 Conclusion

Le consensus de preuve d'utilisation permet d'encourager les participants d'un réseau privé à échanger leur monnaie et à partager leurs données personnelles. Le modèle décrit dans la section 3.2 est une alternative viable aux modèles actuels d'échanges de données, et la preuve d'utilisation propose une solution pour créer un environnement fiable de partage de données personnelles centré sur l'utilisateur. La première version de la preuve d'utilisation est actuellement en librement accessible [1].

Nos travaux ont été mis en valeur par une participation à une conférence internationale [49] ainsi que par une publication dans un journal international [50].

1. https://github.com/Pikciochain/Proof_of_Usage

IV

Preuve d'expérience

Après plus de douze ans de services, Bitcoin reste la crypto-monnaie la plus utilisée et la plus influente sur le marché des crypto-monnaies. Comme vu dans le chapitre 2, à chaque bloc, les mineurs sont mis en compétition pour résoudre un problème mathématique nécessitant un nombre conséquent de calculs. Le premier mineur qui résout le problème, peut créer le bloc suivant et gagner la récompense, le travail réalisé par les autres mineurs est perdu et l'énergie qu'ils ont consommée est gaspillée. Un autre problème que présente le réseau Bitcoin est qu'il récompense majoritairement les utilisateurs ayant le plus de puissance de calcul et pas les utilisateurs qui sont là depuis plus longtemps et qui ont permis au réseau de se développer quand il était moins connu. Ces utilisateurs, sans qui le Bitcoin ne serait pas ce qu'il est aujourd'hui, se sont faits totalement remplacer lorsque le réseau a commencé à être rentable et que d'autres utilisateurs sont arrivés avec une puissance de calcul bien plus conséquente. Dans ce chapitre, nous introduisons le protocole de preuve d'expérience (Proof of Experience, PoE en anglais), un algorithme de consensus pensé pour inciter les utilisateurs d'une blockchain publique à être fidèle au réseau. La preuve d'expérience propose de modifier le consensus de Bitcoin afin que la valeur du hash recherchée (la cible) devienne propre à chaque mineur. La preuve d'expérience permet à un mineur d'augmenter la valeur de sa cible (et donc rendre son problème plus simple) en prouvant aux autres qu'il a fourni un travail sur les blocs précédents.

4.1 Introduction

Dans le chapitre précédent, nous avons présenté la preuve d'utilisation, un consensus visant à récompenser les utilisateurs qui réalisent des transactions sur un réseau au travers d'une blockchain semi-privée. Dans ce chapitre, nous proposons la preuve d'expérience, un consensus incitant les utilisateurs d'une blockchain publique à la loyauté tout en diminuant la gaspillage d'énergie engendré par le consensus que propose le réseau Bitcoin.

Pour ce faire, les utilisateurs travaillant pour l'ajout de blocs à la chaîne obtiennent de l'expérience, plus un utilisateur a d'expérience, plus il a de chance d'obtenir une récompense.

Pour rappel, lorsqu'un mineur ajoute un bloc à la chaîne Bitcoin, le travail que les autres mineurs ont réalisé pour ajouter ce bloc est perdu et l'énergie qu'ils ont investie est gaspillée. La preuve d'expérience propose de réutiliser cette énergie en la convertissant en expérience afin de récompenser indirectement les mineurs pour leur travail, diminuant à l'occasion le gaspillage d'énergie.

De précédents travaux proposent de réutiliser l'énergie investie dans le consensus du Bitcoin. Par exemple, le réseau Permacoin [55], basé sur un consensus utilisant une preuve de récupération (Proof of Retrievability [42]) propose d'utiliser cette consommation d'énergie afin de maintenir un stockage distribué de données d'archives. D'autres blockchains comme Foldingcoin [11], Golem [12] ou Gridcoin [13] proposent un consensus récompensant un utilisateur en fonction de la quantité de calculs scientifiques qu'il a effectués (pour des problèmes d'optimisation notamment). Nous pouvons également citer le consensus de preuve d'apprentissage profond (Proof of Deep Learning [9]) qui utilise le travail réalisé par les mineurs pour entraîner un réseau de neurones [32].

D'autres travaux proposent de récompenser la loyauté des mineurs d'une blockchain publique. C'est le cas notamment de Peercoin [44] basé sur une preuve d'enjeu, dans laquelle la fonction de choix aléatoire est pondérée par « l'âge » des pièces des utilisateurs. Un mineur est sélectionné en fonction d'un nombre résultant du produit du nombre de pièces qu'il possède et du nombre de jours depuis lesquels il les possèdent. Ainsi, plus un utilisateur a de pièces et plus elles sont anciennes, plus il aura de chance d'être sélectionné pour enregistrer le bloc suivant. Seules les pièces qui n'ont pas été dépensées depuis au moins 30 jours peuvent être sélectionnées pour ajouter le bloc suivant. Peercoin incite donc ses utilisateurs à stocker des pièces afin d'être récompensés. Il en résulte un modèle dans lequel les utilisateurs sont encouragés à stocker leurs pièces plutôt qu'à les dépenser.

4.2 Motivations

Actuellement, avec le consensus que propose Bitcoin, plus un mineur a une capacité de calcul importante plus il a de chance d'enregistrer un bloc et donc plus il est récompensé. Considérons un cas, dans lequel un mineur avec une capacité de calcul modeste travaille sur le réseau depuis sa création. Ce mineur contribue au lancement, à la stabilisation et au développement du réseau, pourtant, lorsqu'il commence à être intéressant et devenir rentable, des mineurs avec une puissance de calcul importante arrivent sur le réseau et le mineur qui était là depuis le début ne peut plus prétendre à enregistrer des blocs, il est remplacé par les nouveaux arrivants et la loyauté des utilisateurs de départ n'est pas récompensée. Actuellement, de nombreux réseaux blockchain perdent leurs utilisateurs de départ malgré qu'ils soient généralement à l'origine du développement du réseau.

Le protocole de preuve d'expérience vise à réduire l'influence des mineurs qui arrivent

sur le réseau tardivement. Pour ce faire, la preuve d'expérience propose de récompenser les utilisateurs en fonction de leur puissance de calcul pondérée par le travail que l'utilisateur a déjà fourni sur le réseau (i.e. l'expérience). Intuitivement, plus un utilisateur a travaillé pour le réseau, plus le travail qui lui est demandé est facile. Autrement dit, plus un utilisateur est là depuis longtemps (plus il est loyal) plus il aura de chance d'enregistrer un bloc et donc plus il sera récompensé.

La preuve de travail met en concurrence les mineurs en leur demandant la solution à un problème mathématique nécessitant un nombre important de calculs et seul le premier à résoudre le problème gagne la récompense. L'énergie consommée par le calcul des autres mineurs est gaspillée. Le protocole de preuve d'expérience propose de convertir l'énergie gaspillée en expérience afin de réduire la difficulté du problème demandé par la suite à l'utilisateur.

4.3 Protocol

Le protocole de preuve d'expérience est défini à partir du consensus utilisé par le réseau Bitcoin. En effet, notre modèle utilise le même modèle de transactions (basé sur les sorties de transactions non dépensées ou UTXO), le même système de signatures (basé sur les courbes elliptiques [56]) ainsi que la même preuve de travail que le réseau Bitcoin. Pour rappel, la preuve de travail du réseau Bitcoin consiste à trouver un nombre entier (de 32 bits) appelé nonce telle que la concaténation de l'en-tête du bloc (i.e. le bloc sans la liste des transactions) avec le nonce produisent un hash inférieur ou égal à un hash cible (appelé cible). La cible est un nombre entier composée de 256 bits, connu de l'ensemble des utilisateurs du réseau, qui est réajusté tous les 2016 blocs en fonction de la capacité de calcul totale du réseau. Plus la capacité de calcul augmente, plus la cible diminue et inversement. La difficulté de trouver un nonce valide dépend de la valeur de la cible, plus elle est petite, plus la difficulté est élevée.

Noter que, comme dans le réseau Bitcoin, tous les hashs obtenus sont le résultat de la fonction de hachage SHA-256. À noter également que le consensus que nous proposons allie preuve de travail (basée sur le problème de nonce de Bitcoin) et preuve d'expérience (preuve qu'un mineur a travaillé sur les blocs précédents). Le terme de travail signifie qu'un mineur fournit de la puissance de calcul afin d'essayer de résoudre le problème du nonce.

Dans le protocole de preuve d'expérience, l'expérience de mineurs est représentée par le travail qu'un mineur a fourni précédemment. Plus un mineur a travaillé, plus la difficulté des blocs futurs sera basse. En réalité, le protocole de preuve d'expérience différencie deux types de cibles : la cible globale et la cible locale. La cible globale correspond au hash cible du réseau Bitcoin, elle correspond à la valeur minimale de la cible locale (i.e. la difficulté maximale). Comme dans le réseau Bitcoin, sa valeur est partagée par l'ensemble des utilisateurs du réseau et elle est recalculée tous les 2016 blocs en fonction de la puissance de calcul totale du réseau. Lorsqu'un mineur commence sur le réseau, sa cible est égale à la cible globale, toutefois, à force de travailler, un mineur peut augmenter la valeur de

sa cible et calculer sa propre cible : elle est appelée cible locale. Pour chaque bloc, chaque mineur calcule sa propre cible locale en fonction du travail qu'il a fourni sur les blocs précédents (même si les hashs qu'il a trouvés sont supérieurs à sa cible locale et à la cible globale).

Dans le protocole de preuve d'expérience, comme dans le protocole Bitcoin, chaque mineur essaie de calculer le nonce du bloc suivant afin de gagner la récompense du bloc. Pour construire le bloc suivant b_i, un mineur m doit réaliser les opérations suivantes :

1. **Calcul de la cible locale :** m calcule sa cible locale t_m en fonction de la cible globale et du travail qu'il a fourni précédemment.
2. **Construction de l'en-tête :** m construit l'en-tête du bloc, comprenant la preuve de travail et le calcul du nonce tel que la concaténation de la preuve de travail avec le nonce produisent un hash inférieur ou égal à t_m. À noter que la preuve de travail n'est pas réalisée sur l'ensemble de l'en-tête mais seulement sur certains champs que nous verrons plus loin.
3. **Sélection des transactions :** m ajoute des transactions au bloc à partir des transactions en attente d'être ajoutées.
4. **Partage du bloc :** m calcule le hash final du bloc et partage le bloc aux nœuds complets du réseau.

Afin de stabiliser le réseau, comme pour Bitcoin, les mineurs travaillent toujours sur la chaîne la plus longue. Quand un bloc est ajouté à la chaîne, il est partagé avec tous les nœuds actifs du réseau.

Dans cette section, nous commençons par présenter la structure des blocs de la chaîne utilisée par le protocole de preuve d'expérience. Ensuite, nous voyons comment un mineur peut calculer sa cible locale et comment il prouve cette valeur aux autres utilisateurs du réseau. Les dernières parties de cette section présentent le système de validation des blocs et décrit le comportement de la chaîne dans des cas particuliers tels que un fork ou au lancement de la chaîne (les premiers blocs).

4.3.1 Structure des blocs

L'ensemble des champs cités dans cette sous-section se reportent aux champs de l'en-tête utilisés dans le réseau Bitcoin, voir section 2.2 pour plus de détails. Les cibles locales et globales enregistrées dans les blocs sont enregistrées après les avoir converties au format Bitcoin (en bits, voir figure 2.6 pour plus de détails sur la conversion), toutefois, par abus de langage, nous parlerons de cible globale et de cible locale (même lorsqu'il s'agit de bits de la cible globale ou de bits de la cible locale).

Dans la blockchain Bitcoin, l'en-tête d'un bloc contient l'ensemble des informations du bloc (à l'exception de la liste des transactions que le bloc contient, mais elle est contenue

par la racine de Merkle). Sur le réseau Bitcoin, le calcul de la preuve de travail est réalisé à partir de l'en-tête du bloc et consiste à trouver un nonce tel que le hash de l'en-tête soit inférieur ou égal à la cible. Dans le protocole de preuve d'expérience, nous distinguons la preuve de travail et l'en-tête. L'en-tête contient la preuve de travail mais la preuve de travail n'est pas réalisée sur l'entièreté de l'en-tête.

En plus des champs que contiennent l'en-tête de bloc du réseau Bitcoin, l'en-tête d'un bloc utilisant le protocole de preuve d'expérience contient une séquence de nonces, l'identifiant du mineur qui calcule le bloc et une cible locale (voir figure 4.1). Ces changements sont nécessaires afin de simplifier le calcul et la vérification de la cible locale. En effet, ces changements permettent à un mineur m_1 de prouver l'expérience qu'il a accumulée sur un ensemble de blocs $B = \{b_1, \ldots, b_n\}$ à un mineur m_2 en fournissant seulement son identifiant (ou sa clé publique) et une séquence de nonces calculés pour chacun des blocs. La séquence de nonces correspond aux meilleurs nonces trouvés pour chaque bloc de B (les nonces tels que la preuve de travail retourne un hash le plus petit possible). L'index des blocs de B étant fixé et connu de tous, le mineur m_2 peut reconstruire les n en-têtes, recalculer les hashs trouvés par m_1 et finalement recalculer la cible locale de m_1.

En Bitcoin, lorsqu'un mineur a testé tous les nonces possibles pour un bloc sans trouver une preuve de travail valide, il peut rapidement créer un nouvel en-tête en modifiant l'horodatage et tester les 2^{32} valeurs de nonces possibles sur le nouvel en-tête. Bien que moins utile (car plus lent), un mineur peut également ajouter, supprimer ou remplacer une transaction de la liste de transactions, modifiant ainsi la racine de Merkle des transactions et donc l'en-tête. En retirant l'horodatage et la racine Merkle de la preuve de travail, le mineur ne peut plus modifier la preuve de travail qu'il calcule (à l'exception du nonce). Les mineurs pourraient donc rapidement se retrouver bloqués après avoir calculé les 2^{32} possibilités (en quelques secondes pour les mineurs les plus importants), c'est pourquoi nous avons choisi de doubler l'espace mémoire utilisé par le nonce et passé de 32 bits à 64 bits. Ainsi le nombre desvaleurs possibles que peuvent prendre un nonce passe de 2^{32} ($\simeq 10^9$) à 2^{64} ($\simeq 10^{19}$).

4.3.2 Cible locale

Dans cette sous-section nous présentons comment le calcul de la cible locale d'un mineur peut être réalisé en fonction du travail précédemment fourni sur le réseau. Afin d'ajouter un bloc, un mineur doit trouver un nonce tel que la concaténation du hash du bloc précédent avec l'identifiant du mineur (dérivé de sa clé publique) et le nonce produit un hash inférieur ou égal à sa cible locale. La cible locale d'un mineur correspond à un ajustement de la cible globale, plus un mineur a travaillé sur les blocs précédents, plus la cible locale est élevée et donc plus la difficulté de trouver un nonce est basse. Quand un mineur partage un bloc, les utilisateurs du réseaux doivent être en mesure de recalculer et vérifier la valeur de sa cible locale. La séquence de nonces contenue dans le bloc correspond aux meilleurs nonces trouvés par le mineur pour un ensemble de blocs donné. Quand un nœud reçoit un bloc, il peut reconstruire les preuves de travail réalisées par le mineur du bloc à partir de la séquence de nonces, l'identifiant du mineur et les hashs des blocs de la chaîne (qu'il possède déjà, car le nœud possède la chaîne). À partir de ces

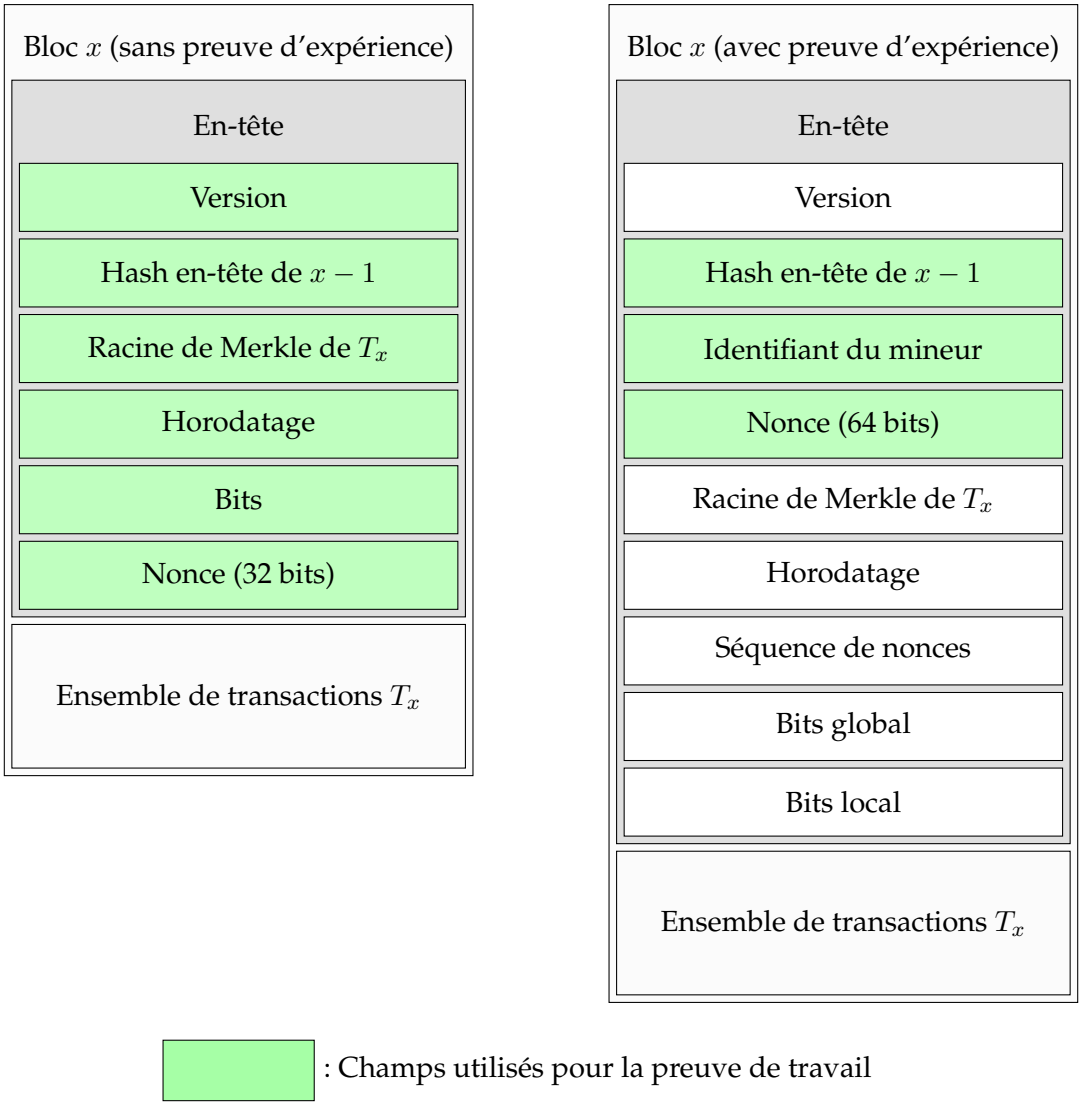


FIGURE 4.1 – *Comparaison entre un bloc de la chaîne Bitcoin (à gauche) et un bloc utilisant la preuve d'expérience (à droite). Les champs en vert correspondent aux champs utilisés pour la preuve de travail.*

preuves de travail, le nœud peut estimer le travail réalisé par le mineur et recalculer sa cible locale.

Idéalement, afin d'estimer au mieux le travail réalisé par un mineur, le calcul de la cible locale devrait considérer l'ensemble des blocs depuis la création de la chaîne. Le problème étant que le temps de calcul augmenterait avec la taille de la chaîne, ce qui n'est pas envisageable sur le long terme. De plus, la séquence de nonces (qui est contenue dans les blocs) deviendrait trop grande et les blocs utiliseraient de plus en plus d'espace mémoire. Afin de limiter le temps de calcul et la taille des blocs, nous ne considérons pas l'ensemble de la chaîne mais seulement un échantillon de blocs parmi les précédents.

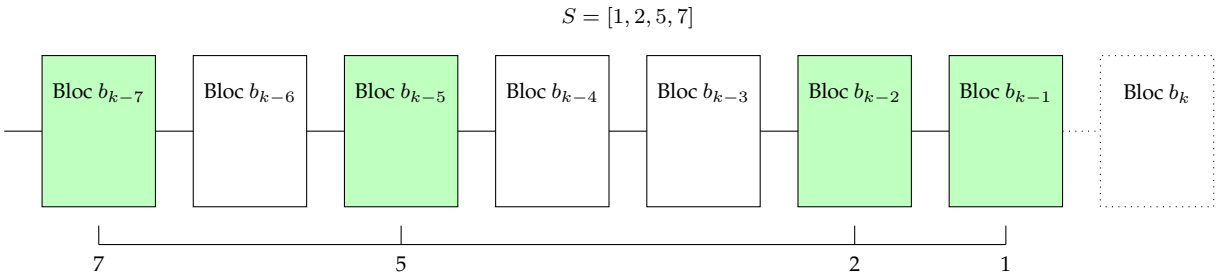


FIGURE 4.2 – *Exemple d'échantillon $S = [1, 2, 5, 7]$. Les blocs en vert correspondent aux blocs pris en compte par la fonction de calcul de la cible locale du bloc b_k.*

Échantillon de blocs

L'échantillon de blocs correspond à une séquence de blocs que les nœuds considèrent pour calculer la cible locale d'un mineur pour un bloc donné (à la fois pour calculer sa cible locale et pour vérifier celle des autres). Considérons une séquence d'entiers positifs $S = [s_1, s_2, \ldots, s_n]$ (avec $s_1 < s_2 < \cdots < s_n$) et un mineur m_1 qui partage le bloc b_i, le $i^{\text{ème}}$ bloc de la chaîne. Afin de calculer la cible locale de m_1 pour b_i, un mineur m_2 (possiblement m_1) considère le travail de m_1 réalisé sur l'ensemble des blocs $B = \{b_{i-s_j} \ : \ s_j \in S\}$ (voir exemple figure 4.2). Avec un tel système, un nœud peut estimer le travail réalisé par un mineur sur les s_n derniers blocs en vérifiant seulement n blocs.

Les éléments et la taille de l'échantillon peuvent être fixes et connus par tous les nœuds du réseau ou alors ils peuvent être calculés dynamiquement, dans ce cas, c'est la fonction qui les calcule qui doit être déterministe et connue par l'ensemble des nœuds. Un exemple de séquence fixe : $S = [x \ : \ x$ est un nombre impair $\wedge \ x \in [\![1, 99]\!]]$. Dans cet exemple, la taille et les éléments de S sont fixes et connus de tous, les nœuds peuvent donc vérifier la cible locale d'un bloc sans calculer S au préalable. Maintenant considérons la séquence $S = [x \ : \ x = i - 100k, \forall k \in [\![1, i/100]\!]]$, avec i le numéro du bloc courant. Les éléments et la taille de cette séquence ne sont pas fixes, donc un nœud doit recalculer S à chaque fois qu'il veut calculer une cible locale.

Considérons le bloc b_i, le $i^{\text{ème}}$ bloc de la chaîne. La taille de la séquence $|S|$ correspond au nombre de blocs à évaluer pour calculer la cible locale de b_i et le numéro du bloc courant moins la plus grande valeur de la séquence $(i - s_n)$ correspond au numéro du plus ancien bloc évalué. Ainsi, un mineur qui commence à travailler pour le réseau atteindra sa quantité maximale d'expérience après avoir travaillé sur les s_n blocs suivants. Nous présentons les caractéristiques que doivent présenter l'échantillon choisi dans la sous-section 4.4.1.

Calcul de la cible locale

La fonction de calcul de la cible locale est une fonction déterministe, accessible à l'ensemble des nœuds du réseau. Cette fonction est utilisée à la fois par les mineurs pour calculer leur cible locale mais également par les nœuds complets pour vérifier que le mineur d'un bloc ait correctement calculé sa cible locale (vérifier qu'il n'a pas essayé de tricher en augmentant sa cible locale).

Précédemment, nous avons vu que chaque mineur possède sa propre cible locale pour un même bloc b_i en fonction de la quantité de travail qu'il a fourni à la chaîne précédemment. Plus un mineur a fourni de travail, moins il devra travailler pour les blocs suivants. De nombreuses fonctions peuvent permettre de calculer la cible locale d'un mineur. De la définition de cette fonction dépend le comportement des utilisateurs et la stabilité du réseau. Nous proposons des fonctions basées sur une cible globale du réseau (comme la cible Bitcoin) qui augmente pour chaque mineur et fonction de la quantité de travail qu'il a déjà fournie. Dans cette sous-section, nous présentons les caractéristiques que nous pensons primordiales pour une fonction de calcul de la cible locale.

La fonction de calcul de la cible locale t_m d'un bloc b_i pour un mineur m en fonction du travail qu'il a fourni sur les blocs $B = \{b_j \ : \ j = i - s_k, s_k \in S\}$ (avec S un échantillon, une séquence d'entiers positifs) est décrite par les entrées/sorties suivantes :

> ***Entrée:*** Une séquence d'entiers positifs $S = \{s_1, s_2, \ldots, s_n\}$, une séquence de nonces $N = \{x_1, x_2, \ldots, x_n\}$, l'identifiant du mineur m et un bloc b_i.
>
> ***Sortie:*** La cible locale t_m (nombre entier codé sur 256 bits) du mineur m pour le bloc b_i.

À noter que la séquence de nonces N correspond aux meilleurs nonces trouvés par le mineur m (les nonces qui produisent les hashs les plus petits). La séquence de nonce N est ordonnée de façon à ce que le nonce $x_k \in N$ corresponde au meilleur nonce trouvé par le mineur m pour la preuve de travail du bloc b_{i-s_k}.

Afin de calculer la cible locale d'un bloc, un mineur m' (possiblement $m' = m$) doit recréer et recalculer les preuves de travail calculées par m pour chaque bloc de B. Ainsi, $\forall s_k \in S$, m' calcule la preuve de travail de m pour le bloc b_{i-s_j} en calculant le hash de la concaténation des éléments suivants :

- le hash du bloc précédent b_{i-s_j-1} (accessible directement dans la chaîne),
- l'identifiant du mineur m (contenu dans le bloc b_i),
- le nonce x_{s_j} de la séquence de nonces du b_i.

Ainsi, en seulement $|B|(= n)$ exécution de la fonction de hachage, m' peut calculer et évaluer le travail fourni par m pour les blocs de B. Le mineur m' peut donc recalculer la cible locale de m pour le bloc b_i en seulement $|B|$ hashs.

À noter que la chaîne peut rapidement devenir vulnérable à une attaque de double dépense si le calcul de la cible locale se base uniquement sur le travail fourni par les mineurs. En effet, sur le long terme, le mineur avec la puissance de calcul la plus importante finira par avoir la cible locale la plus grande (i.e. la difficulté la plus basse du réseau pour ajouter un bloc) augmentant ainsi son influence sur le réseau. Si ce mineur est malveillant, il peut créer un fork local, calculer des blocs pour créer une chaîne plus grande que la

chaîne courante et forcer les autres nœuds à passer sur sa chaîne. Dans ce cas là, les blocs créés depuis le fork sur l'ancienne chaîne deviennent orphelins et sont supprimés. En plus de réussir une attaque de double dépense, le mineur malveillant supprime le travail des mineurs ayant travaillé sur les blocs orphelins. Il diminue ainsi l'expérience des autres mineurs tout en augmentant la sienne, il augmente donc encore plus son influence sur le réseau rendant une attaque de double dépense encore plus simple à réaliser pour lui.

Afin de prémunir le réseau d'une attaque de double dépense, d'autres paramètres sont nécessaires à prendre en compte dans le calcul de la cible locale. Par exemple, on peut envisager de donner une pénalité (une diminution de l'expérience) importante à un mineur qui a ajouté les x derniers blocs. Il est possible également de regarder les x derniers blocs et donner une pénalité aux mineurs en fonction du nombre de blocs qu'un mineur a ajouté. Par exemple, si un mineur m_1 a enregistré cinq blocs parmi les deux-cents derniers et un autre mineur m_2 en a enregistré cent dans le même intervalle, la fonction de calcul de la cible locale va retirer un peu d'expérience à m_1 et beaucoup d'expérience à m_2. Cette pénalité permet de prémunir le réseau d'une attaque de double dépense (en diminuant l'influence des mineurs ayant une puissance de calcul importante) et permet d'éviter d'augmenter l'écart de puissance entre les mineurs.

Comme dit précédemment, il y a de nombreuses fonctions permettant de calculer la cible locale et chacune d'entre elles aura un impact différent sur le réseau, toutefois, il semble important que la cible locale soit toujours supérieure ou égale à la cible globale (sinon le mineur a plus intérêt à recréer un compte pour partir sans expérience qu'à continuer à travailler avec son compte actuel). De plus, dans le réseau Bitcoin, il existe une cible maximum correspondant à la plus grande valeur que la cible globale peut prendre. La cible maximum permet de toujours demander un minimum de travail pour l'enregistrement d'un bloc, même dans le cas où la puissance de calcul totale du réseau viendrait à diminuer significativement. Il est donc important que la cible locale soit toujours inférieure ou égale à la cible maximum. Nous devons donc définir des fonctions qui calculent une cible locale bornée inférieurement par la cible globale et supérieurement par la cible maximum.

Dans la sous-section 4.4.2 nous présentons les fonctions qui nous semblent les plus pertinentes pour un réseau utilisant la preuve d'expérience.

4.3.3 Création de la séquence de nonces

Considérons la séquence d'ensembles positifs $S = \{s_1, s_2, \ldots, s_n\}$ (avec $s_1 < s_2 < \cdots < s_n$) à partir de laquelle l'échantillon de blocs B est généré.

Propriété 1 : En considérant qu'un mineur m enregistre localement les nonces qui produisent le hash le plus faible pour chaque bloc entre le bloc courant b_i et le bloc b_{i-s_n}, alors m peut créer la séquence de nonces du bloc b_i en temps constant (en $O(|S|)$).

Démonstration. En considérant que le mineur m enregistre ses meilleurs nonces dans un tableau de taille s_n. Pour chaque $j^{\text{ème}}$ bloc b_j, m enregistre le nonce correspondant dans

le tableau dans la case j modulo s_n. Donc, $\forall k \in [\![1, s_n]\!]$, m peut accéder au meilleur nonce qu'il a calculé pour le bloc b_{i-k} en consultant son tableau à la case $i - k$ modulo s_n. Ainsi, le mineur m peut créer sa séquence de nonces pour le bloc $i + 1$ en consultant $|S|$ fois son tableau. ■

4.3.4 Vérification des blocs

Soit m', un nœud du réseau recevant le bloc b_i construit par le mineur m. Afin de vérifier la validité du bloc b_i, m' vérifie que les conditions suivantes sont respectées (de nombreuses conditions sont les mêmes que pour les blocs du réseau Bitcoin) :

- La version du consensus utilisée correspond à la version actuelle.
- Le hash du bloc proposé correspond bien au hash de l'en-tête du bloc précédent.
- L'identifiant correspond à l'identifiant du mineur m.
- Les transactions enregistrées dans le bloc sont valides, elles sont toutes signées par l'émetteur de la transaction et aucune UTXO d'entre elles n'a déjà été utilisée.
- La racine de l'arbre de Merkle de l'en-tête correspond bien à la racine obtenue à partir des transactions du bloc.
- L'horodatage respecte bien les contraintes d'antériorité à deux heures dans le futur et de postériorité à l'horodatage médian des 11 derniers blocs.
- La valeur de la cible globale utilisée est la valeur correspondante à la cible globale actuelle du réseau.
- La valeur de la cible locale à été correctement calculée, m' exécute la fonction de calcul de cible locale et vérifie que le résultat obtenue est le même que celui inscrit dans le bloc.
- Le résultat de la fonction de hachage SHA-256 de la concaténation du hash précédent, de l'identifiant du mineur et du nonce est inférieur à la cible locale.

Si toutes ces conditions sont respectées alors le bloc est considéré comme valide, m' peut l'ajouter à sa chaîne.

La différence de temps de vérification d'un bloc Bitcoin avec ou sans la preuve d'expérience correspond à la durée d'exécution de la fonction calculant la cible locale du bloc. Cette durée dépend de la fonction choisie.

4.3.5 Cas particuliers

Dans cette partie, nous voyons comment la preuve d'expérience peut gérer les deux cas particuliers suivant : le début de la chaîne et le fork.

Soit $S = \{s_1, s_2, \ldots, s_n\}$ (avec $s_1 < s_2 < \cdots < s_n$) une séquence d'entiers positif. Au lancement de la chaîne (avant le $s_n{}^{\text{ème}}$ bloc) certains blocs nécessaires au calcul de la cible locale n'existent pas. Dans ce cas, le calcul de la cible n'est exécuté qu'avec les blocs existants. Ainsi, les mineurs qui travaillent dès le premier bloc profiteront pleinement de leur expérience à partir du $(s_n + 1)^{\text{ème}}$ bloc de la chaîne.

Comme dans le réseau Bitcoin, un fork involontaire peut se créer dans le protocole de preuve d'expérience lorsque deux mineurs ou plus trouvent et partagent un bloc dans le même laps de temps. Toutefois, en considérant que les nœuds ne considèrent que la chaîne la plus longue, lorsque l'une des deux chaînes créées devient plus longue que l'autre, elle devient privilégiée par le réseau au détriment de la chaîne plus courte qui est abandonnée et dont les blocs deviennent orphelins. Pour le réseau, les blocs orphelins n'existent pas, le travail qu'un mineur à fourni pour un bloc devenu par la suite orphelin ne peut donc pas être vérifié et ne peut donc pas être utilisé pour le calcul de ses futures cibles locales. Le travail que fournissent donc les mineurs sur des blocs orphelins est malheureusement perdu.

4.4 Discussions

Dans la première partie de cette sous-section, nous présentons les caractéristiques qui nous semblent nécessaires afin de créer l'échantillon de bloc le plus représentatif possible, puis, à travers quelques exemples nous montrons l'influence de l'échantillon choisi sur le poids de la chaîne. Dans la seconde partie, nous présentons les fonctions que nous avons implémentées et testées afin d'analyser leur influence sur le réseau.

4.4.1 Échantillon de blocs

Comme vu précédemment, la taille de l'échantillon et les valeurs qu'il contient peuvent être fixes ou recalculées dynamiquement. Un échantillon fixe connu de tous les nœuds permet de simplifier le calcul de la cible locale. Nous analyserons principalement des échantillons de tailles et de valeurs fixes. L'échantillon choisi doit permettre d'avantager un mineur qui travaille continuellement par rapport à un mineur qui travaille moins régulièrement. Un mauvais exemple d'échantillon serait par exemple de prendre la séquence d'entiers suivante : $S = [10k \ : \ \forall k \in [\![1, n]\!]]$ (avec n un entier positif arbitraire). En effet, si l'on considère un mineur m_1 travaillant continuellement sur tous les blocs et un mineur m_2 travaillant seulement un bloc sur dix, alors, tous les dix blocs, m_1 et m_2 auront leur expérience maximale pour ajouter le bloc suivant. Dans ce cas là, à puissance de calcul égale, m_1 et m_2 auront autant de chance d'enregistrer le bloc suivant, l'échan-

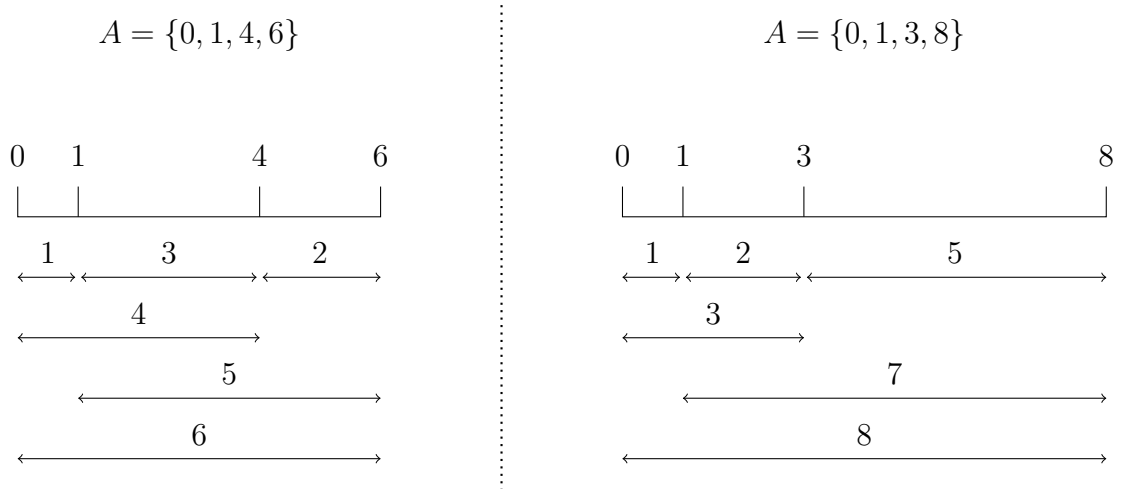


FIGURE 4.3 – *Deux exemples de règles de Golomb. À gauche, une règle de Golomb d'ordre 4 et de taille 6, à droite une règle de Golomb d'ordre 4 et de taille 8. À noter que la règle de gauche est optimale (i.e. sa taille est minimale étant donné son ordre) mais ce critère ne sera pas un critère nécessaire pour définir les règles que nous utilisons en tant qu'échantillon pour la preuve d'expérience.*

tillon ne permet pas de pénaliser un mineur qui travaille seulement un bloc sur dix par rapport à un autre qui travaille continuellement. Afin de vérifier qu'un mineur travaille continuellement, l'échantillon le plus précis serait de regarder l'ensemble des blocs précédents, toutefois, cet échantillon serait trop coûteux à la fois en terme de temps de calcul de la cible locale et en terme d'espace mémoire (car la séquence de nonces est enregistrée sur la chaîne).

Les échantillons basés sur des règles de Golomb [64, 68] permettent de vérifier efficacement si un mineur travaille continuellement pour le réseau ou non. Une règle de Golomb est un ensemble de marques réparti le long d'une règle imaginaire sur des valeurs entières telles que deux paires de marques ne soient jamais à la même distance (voir exemple figure 4.3). Le nombre de marques est appelé ordre et la taille d'une règle de Golomb correspond à la distance séparant les deux marques les plus éloignées. Formellement, soit $A = a_1, a_2, \ldots, a_n$ (avec $a_1 < a_2 < \cdots < a_n$) un ensemble d'entiers positifs, A est une règle de Golomb d'ordre n et de taille $a_n - a_1$ si et seulement si :

$$\forall i, j, k, l \in \{1, 2, \ldots, n\}\ a_i - a_j = a_k - a_l \wedge i \neq j \iff (i = k) \wedge (j = l).$$

Propriété 2 : Soit $S = [s_1, s_2, \ldots, s_n]$ une séquence d'entiers positifs telle que S est une règle de Golomb, pour tout couple de blocs (b_i, b_j) (avec $i < j$), il existe au plus un bloc b_c qui prend en compte à la fois le travail réalisé sur le bloc b_i et celui réalisé sur le bloc b_j dans le calcul de sa cible locale.

Démonstration. Considérons que b_i et b_j sont utilisés dans le calcul de la cible locale du bloc b_c. Donc il existe $s_k, s_l \in S$ tel que $c - s_l = i$ et $c - s_k = j$. Considérons maintenant que b_i et b_j sont réutilisés pour calculer la cible locale d'un autre bloc b_{c+d} (avec $d > 0$). Alors, il existe $s_{k'}, s_{l'} \in S$ tel que $c + d - s_{l'} = i$ et $c + d - s_{k'} = j$. En combinant nos équations, on a $s_k = s_{k'} - d$ et $s_l = s_{l'} - d$ et donc $d = s_{k'} - s_k = s_{l'} - s_l$ (avec $s_k \neq s_{k'}$) tel que $s_k \neq s_l$ et $s_{k'} \neq s_{l'}$. Conclusion : S n'est pas une règle de Golomb, contradiction avec

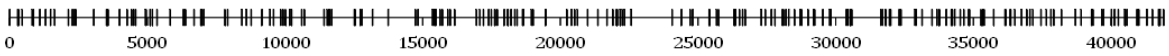


FIGURE 4.4 – *Règle de Golomb d'ordre 212 et de taille 41 911.*

l'hypothèse de départ. ■

En utilisant une règle de Golomb en tant qu'échantillon, par la propriété 2, pour tout couple de blocs (b_i, b_j) (avec $i \neq j$), il existe au plus un bloc qui prend en compte le travail du mineur sur le bloc b_i et sur le bloc b_j pour le calcul de sa cible locale. Avec un tel échantillon, lorsqu'un mineur essaie de tricher en travaillant seulement sur certains blocs, il sera rapidement pénalisé car tout couple de blocs sur lequel il travaille ne sera utilisé au plus qu'une fois dans le calcul d'une cible locale.

Soit S un échantillon tel que $A = s \ : \ s \in S$ est une règle de Golomb. En considérant le bloc b_i, le $i^{\text{ème}}$ de la chaîne, l'ordre de la règle de Golomb (i.e. $|A| = |S|$) correspond au nombre de blocs pris en compte dans le calcul de la cible locale de b_i et le numéro du bloc moins la plus grande valeur de la règle (i.e. $i - s_n$) correspond au numéro du bloc le plus ancien pris en compte dans le calcul. En considérant qu'un bloc est ajouté toutes les dix minutes en moyenne, l'échantillon S permet d'estimer le travail d'un mineur sur les $10 \times s_n$ minutes en $|S|$ hashs.

Il existe de nombreuses règles de Golomb de tout ordre et de toute taille. Analysons un exemple concret avec la règle de Golomb A_{212} (voir figure 4.4), une règle d'ordre 212 et de taille $41\ 911$ (accessible sur la page internet de Lloyd Miller [1]). Cette règle est décrite par les éléments suivants :

$A_{212} = \{0,\ 6,\ 298,\ 333,\ 458,\ 823,\ 868,\ 1096,\ 1305,\ 1487,\ 1637,\ 2136,\ 2279,\ 2360,\ 2413,$
$3057,\ 3520,\ 3542,\ 3575,\ 3990,\ 4268,\ 4433,\ 4507,\ 4578,\ 4938,\ 4957,\ 5104,\ 5186,\ 5355,\ 5834,\ 6312,$
$6387,\ 6699,\ 6943,\ 6981,\ 7048,\ 7821,\ 7919,\ 7937,\ 8438,\ 8600,\ 8774,\ 9162,\ 9457,\ 9586,\ 9846,\ 9932,$
$10021,\ 10167,\ 10232,\ 10240,\ 10607,\ 10706,\ 11420,\ 11538,\ 11592,\ 11615,\ 11695,\ 12522,\ 12556,$
$12732,\ 12789,\ 13179,\ 13748,\ 13759,\ 14736,\ 14830,\ 15362,\ 15430,\ 15454,\ 15455,\ 15481,\ 15633,$
$15718,\ 15914,\ 16002,\ 16163,\ 16953,\ 17077,\ 17208,\ 17397,\ 17493,\ 17615,\ 17631,\ 17710,\ 17962,$
$18088,\ 18102,\ 18208,\ 18347,\ 18430,\ 18630,\ 18662,\ 18952,\ 19012,\ 19458,\ 19468,\ 20240,\ 20403,$
$20520,\ 20624,\ 20633,\ 20992,\ 21364,\ 21693,\ 21908,\ 22038,\ 22059,\ 22066,\ 22173,\ 22251,\ 22327,$
$22574,\ 24059,\ 24397,\ 24399,\ 24715,\ 24805,\ 25401,\ 25421,\ 25675,\ 25745,\ 26316,\ 26320,\ 26363,$
$26504,\ 26619,\ 26716,\ 27299,\ 27443,\ 27458,\ 27663,\ 27795,\ 28065,\ 28101,\ 28192,\ 28292,\ 28506,$
$28712,\ 28768,\ 28999,\ 29193,\ 29198,\ 29235,\ 29532,\ 29785,\ 29825,\ 30388,\ 30437,\ 30524,\ 30585,$
$31636,\ 31649,\ 31695,\ 31804,\ 31816,\ 31953,\ 31997,\ 32308,\ 32371,\ 32881,\ 32922,\ 33138,\ 33501,$
$33672,\ 33744,\ 33773,\ 34013,\ 34199,\ 34238,\ 34322,\ 34524,\ 34652,\ 34829,\ 34859,\ 35280,\ 35342,$
$35390,\ 35741,\ 36152,\ 36183,\ 36339,\ 36356,\ 36509,\ 36751,\ 36941,\ 37052,\ 37164,\ 37455,\ 37653,$
$37817,\ 37919,\ 38198,\ 38707,\ 38915,\ 39291,\ 39343,\ 39662,\ 39720,\ 40016,\ 40124,\ 40266,\ 40504,$
$40507,\ 40573,\ 40943,\ 41007,\ 41301,\ 41502,\ 41728,\ 41778,\ 41911\}.$

En prenant un échantillon $S_{212} = [s_i + 1 \ : \ s_i \in A_{212}]$ (avec $s_1 < s_2 <, \ldots, < s_{212}$) et en considérant qu'un bloc est enregistré en moyenne toutes les dix minutes, l'échantillon S_{212} permet d'estimer le travail réalisé par un mineur sur la chaîne lors des 292 derniers

1. http://www3.telus.net/millerlf/g3-records.html

jours (approximativement dix mois) en évaluant seulement 212 blocs (i.e. en calculant 212 hashs).

Comme nous avons vu, la taille d'un bloc, correspond à la taille d'un bloc de la chaîne Bitcoin auquel s'ajoute la séquence de nonces, l'identifiant du mineur ainsi que la cible locale. En Mars 2021, la taille moyenne d'un bloc Bitcoin est de 1.315 Mo [2]. La séquence de nonces est constitué de 212 nonces, chacun d'entre eux correspondant à un entier codé sur 64 bits (8 octets), pour un poids total de 1 696 octets. L'identifiant du mineur est codé sur 256 bits (32 octets) et la version simplifiée de la cible locale (le bits, voir figure 2.6) est codée sur 32 bits (4 octets). Au total, en utilisant la preuve d'expérience avec l'échantillon S_{212}, 1 732 octets sont ajoutées à la taille d'un bloc (soit une augmentation de 0.13%). Le poids moyen des blocs de la chaîne Bitcoin augmentant au fil du temps, on peut réaliser le même calcul en considérant la chaîne Bitcoin depuis le bloc genèse. Les 680 247 blocs de la chaîne Bitcoin pèsent actuellement 341.16 Go (le 20 Avril 2021 [3]). Si la chaîne Bitcoin utilisait la preuve d'expérience depuis son bloc genèse, le poids total de la chaîne aurait été augmenté de 1.18 Go (soit une augmentation de 0.52% sur les 12 premières années de la chaîne).

Pour résumer, une blockchain qui utilise la preuve d'expérience avec l'échantillon S_{212} augmente le temps de calcul d'un bloc de 212 hashs (négligeable par rapport aux $\simeq 1,7 \times 10^{17}$ hashs par seconde que réalise le réseau Bitcoin [7]) et le poids total de la chaîne de 0.52%.

4.4.2 Calcul de la cible locale

Dans cette sous-section, nous proposons quelques fonctions de calcul de la cible locale permettant de réduire l'influence d'un mineur rejoignant le réseau avec une puissance de calcul importante. Afin de tester l'efficacité de nos fonctions, nous avons implémenté une version simplifiée du protocole Bitcoin. Cette version est sans transaction, sans signature asymétrique des blocs et avec une faible tolérance aux pannes. Ce protocole à été implémenté avec le langage Python 3 [4] et fonctionne sur quatre Raspberry Pi 2 modèle B [5], un ordinateur portable (Asus R511L [6]) ainsi qu'une carte graphie (MSI Geforce GTX 1080 Ti GAMING X 11G [7]), tous reliés sur réseau local par un switch (Cisco Catalyst 2950 [8]). Tous ces appareils possèdent l'ensemble de la chaîne et tentent d'ajouter des blocs à la chaîne. Chacun de ces six appareils simule donc à la fois un nœud complet et un mineur sur le réseau.

2. https://ycharts.com/indicators/bitcoin_average_block_size
3. https://ycharts.com/indicators/bitcoin_blockchain_size
4. https://www.python.org/download/releases/3.0/
5. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
6. https://www.topachat.com/pages/detail2_cat_est_ordinateurs_puis_rubrique_est_wport_puis_ref_est_in10090896.html#fiche-technique
7. https://fr.msi.com/Graphics-Card/GeForc-GTX-1080-Ti-GAMING-X-11G/Specification
8. https://www.cisco.com/c/dam/global/fr_fr/assets/documents/pdfs/datasheet/switching/Cat2950_fr_v3.pdf

Afin d'évaluer l'influence d'un mineur rejoignant le réseau avec une puissance de calcul importante, nous avons défini un protocole de test. Le protocole est le suivant :

- Tous les mineurs à l'exception du plus puissant travaillent dès le lancement du réseau, cumulant ainsi de l'expérience.
- Après s_n blocs les mineurs atteignent le maximum d'expérience qu'ils peuvent avoir avec leur puissance de calcul (avec s_n la valeur maximale de l'échantillon utilisé S). Le sixième mineur (ayant la puissance de calcul la plus importante) commence à travailler pour le réseau.

L'objectif de ce protocole est d'observer l'évolution du pourcentage de blocs que rajoute chaque mineur à la chaîne et de voir au bout de combien de blocs le sixième mineur peut influencer le réseau.

Avant de présenter nos fonctions et nos résultats, il nous faut déterminer la puissance de calcul que chacun de nos mineurs possède sur le réseau. Cette puissance est exprimée en pourcentage en fonction de la puissance globale du réseau. Pour ce faire, nous utilisons le protocole suivant : les six mineurs travaillent en même temps sur le réseau avec un consensus sans preuve d'expérience (seulement avec la preuve de travail). Le pourcentage de blocs ajoutés par chaque mineur correspond au pourcentage de sa puissance par rapport à la puissance totale du réseau. Les résultats que nous obtenons sont présentés table 4.1.

Appareil	Pourcentage de blocs enregistrés
Ordinateur portable	34,2
Carte graphique	52,2
Raspberry 1	3,5
Raspberry 2	3,4
Raspberry 3	3,1
Raspberry 4	3,6

TABLE 4.1 – *Pourcentage de blocs enregistrés par chaque mineur du réseau test sans preuve d'expérience (sur 5370 blocs).*

Comme nous pouvions le prévoir, le mineur ayant la puissance de calcul la plus importante est celui simulé par la carte graphique. Dans les tests suivants, la carte graphique sera donc notre sixième mineur et rentrera sur le réseau à partir du $s_n + 1^{\text{ème}}$ blocs.

Comme dit précédemment, de nombreuses fonctions peuvent être utilisées pour calculer la cible locale. Toutefois, nous cherchons une fonction avec laquelle le mineur entrant n'a que très peu d'influence sur le réseau et n'atteint son plein potentiel qu'après avoir travaillé sur s_n blocs (dans notre protocole, à partir du $2s_n^{\text{ème}}$ bloc). Idéalement, les tests devraient être réalisés sur un échantillonnage de taille importante (tel que le S_{212} basé sur

la règle de Golomb A_{212}, présentée figure 4.3) et avec un temps moyen de dix minutes entre chaque bloc (comme pour le réseau Bitcoin). Toutefois, pour des raisons évidentes de temps (plus de vingts mois pour chaque fonction avec l'échantillon S_{212} et dix minutes par bloc), nous utilisons un temps moyen de une minute entre chaque bloc et l'échantillon $S_{61} = [s_i + 1 \ : \ s_i \in A_{61}]$ (avec $s_1 < s_2 <, \ldots, < s_{61}$), un échantillon basé sur la règle de Golomb A_{61} d'ordre 61 et de taille 3134 :

$A_{61} = \{0, 4, 28, 44, 94, 300, 371, 409, 513, 544, 611, 632, 689, 745, 763, 775, 804, 836, 984, 987, 1083, 1105, 1125, 1242, 1251, 1265, 1409, 1442, 1626, 1652, 1662, 1677, 1679, 1749, 1754, 1833, 1928, 1965, 2008, 2118, 2157, 2203, 2211, 2279, 2314, 2476, 2536, 2588, 2601, 2702, 2721, 2766, 2829, 2929, 2935, 2936, 2984, 3018, 3065, 3123, 3134\}$.

Soit t_m la cible locale du mineur m pour le bloc b_i, le $i^{\text{ème}}$ de la chaîne. La première fonction que nous présentons est basée sur un système de bonus/malus. Plus un mineur a travaillé, plus il a de bonus. Le nombre de malus augmente en fonction du nombre de blocs que le mineur a ajouté récemment. La fonction que nous proposons est la suivante :

- Reconstituer les preuves de travail de l'ensemble des blocs correspondants à l'échantillon puis calculer l'ensemble H, l'ensemble des hashs de leur preuve de travail.
- Dans les 256 derniers blocs de la chaîne, pour chaque bloc b_j ajouté par m, ajouté à l'ensemble $Malus$ la valeur $256-(b_i-b_j)$. Par exemple, si m a ajouté le bloc précédent et un autre bloc il y a 102 blocs, nous ajoutons à $Malus$ la valeur 255 (=256-1) et 154 (=256-102). Ainsi, chaque bloc enregistré par un mineur lui coûte un malus, pondéré dynamiquement par l'inverse de son ancienneté. Autrement dit, plus un bloc a été ajouté récemment, plus son poids est important.
- Calculer $Somme = \sum H + \sum Malus$, la somme des hashs de H auquel est ajoutée la somme des malus.
- Calculer $M = Somme/|S|$. À ce stade, nous avons calculé la moyenne du travail réalisé par le mineur m sur l'échantillon S auquel nous avons ajouté un malus en fonction des blocs enregistrés récemment.
- Soit T_{max} et T_g les valeurs respectives de la cible maximum du réseau et de la cible globale actuelle, calculer $t_m = T_g + (T_{max} - M)$. Ce calcul permet d'inverser la moyenne que nous venons de calculer afin que plus un mineur travaille, plus sa cible soit élevée.
- Comme dit précédemment, il est important que la cible calculée soit toujours entre T_{max} et T_g, la fonction retourne donc T_{max} si $t_m > T_{max}$, T_g si $t_m < T_g$ et t_m sinon.

Avec cette fonction, plus un mineur travaille pour le réseau, plus sa cible locale est élevée. Toutefois plus il ajoute de blocs récents plus elle est basse. Ce malus permet d'éviter un emballement dans lequel un mineur avec une puissance de calcul plus importante qu'un autre aurait une cible de plus en plus simple diminuant de plus en plus les chances de l'autre mineur d'ajouter un bloc.

Les comportements des mineurs sur un réseau utilisant la fonction proposée précédemment sont observables figure 4.5.

Dans les 3135 blocs, nous observons une certaine stabilité, l'ordinateur portable enregistre entre 70% et 80% des blocs de la chaîne et chaque raspberry entre 5% et 10%. Lorsque la carte graphique rentre sur le réseau, elle atteint directement son plein potentiel puis le réseau s'équilibre avec les mêmes proportions que pour le réseau sans preuve d'expérience. Ce comportement s'explique par le fait que le bonus donné pour l'expérience des mineurs ne permet pas de compenser le malus qui est donné lorsqu'ils ajoutent un bloc. Ainsi, lorsque la carte graphique commence à miner, tous les mineurs du réseau ont une cible locale égale à la cible globale (exactement comme sur un réseau sans preuve d'expérience).

Pour palier à ce problème, nous avons ajouté une pondération dynamique aux blocs en fonction de leur ancienneté. Cette nouvelle fonction est la même que la précédente à l'exception que pour chaque bloc correspondant à l'élément s_i de l'échantillon S son hash est pondéré par $\lceil log_{10}(s_i) \rceil$. Ainsi, le bloc précédent aura une pondération de 1, alors que le bloc ajouté il y a 2128 blocs aura une pondération de 3. Le comportement des mineurs avec cette nouvelle fonction est présenté figure 4.6.

Dans cette nouvelle fonction, nous observons dans un premier temps que la proportion de blocs enregistrés par l'ordinateur portable augmente de plus en plus au détriment des raspberrys. Ce comportement s'explique par un malus trop peu important par rapport au bonus donné à l'ordinateur portable, il s'agit d'un emballement comme décrit précédemment. Toutefois, avec cette fonction, lorsque la carte graphique arrive sur le réseau (au bloc 3135), elle n'a que très peu de pouvoir sur le réseau, elle doit attendre plus de 1500 blocs avant de commencer à ajouter des blocs et la totalité de son expérience (3135 blocs) pour atteindre son plein potentiel.

De notre point de vue, le comportement des mineurs découlant de la seconde fonction est « meilleure » que celui de la première malgré l'emballement observé dans les 3135 premiers blocs. L'objectif sera donc de tester d'autres fonctions afin de supprimer cet emballement.

4.5 Conclusion

Avec l'ajout de la preuve d'expérience au consensus du réseau Bitcoin, plus un mineur travaille sur le réseau, plus il gagne de l'expérience et plus il gagne de l'expérience, plus la preuve de travail qu'il doit calculer devient simple. Ainsi, plus un mineur travaille pour le réseau, plus il a de chance d'enregistrer un bloc et de gagner la récompense correspondante.

Avec l'utilisation d'échantillons $S = [s_1, \ldots, s_n]$, nous avons vu qu'il est possible d'évaluer rapidement (i.e. en exécutant n fois la fonction de hachage) le travail d'un mineur sur les s_n derniers blocs. L'utilisation d'une règle de Golomb en tant qu'échantillon permet

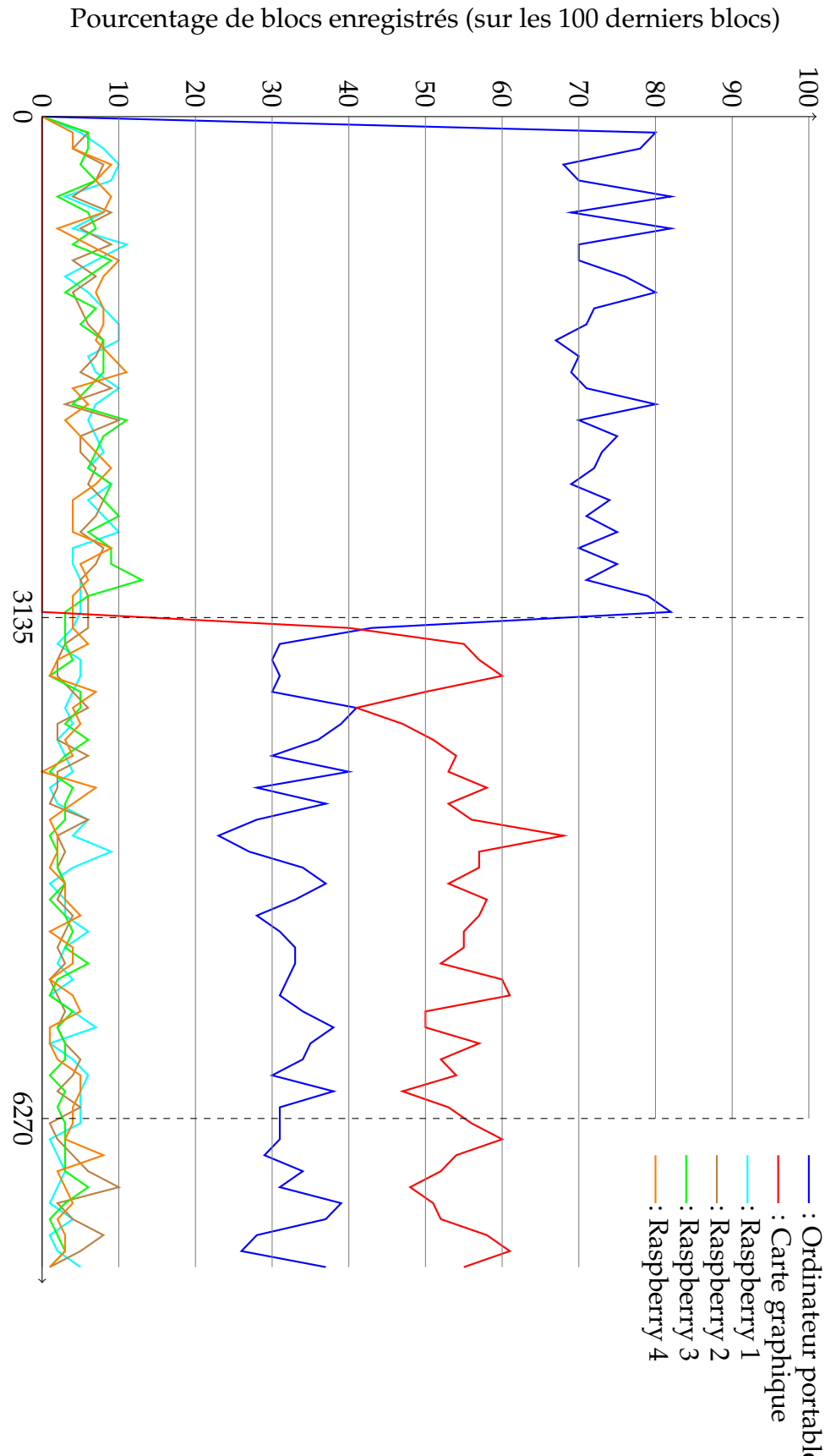


FIGURE 4.5 – *Nombre de blocs moyens enregistrés par chaque mineur au cours de la création de la chaîne (première fonction).*

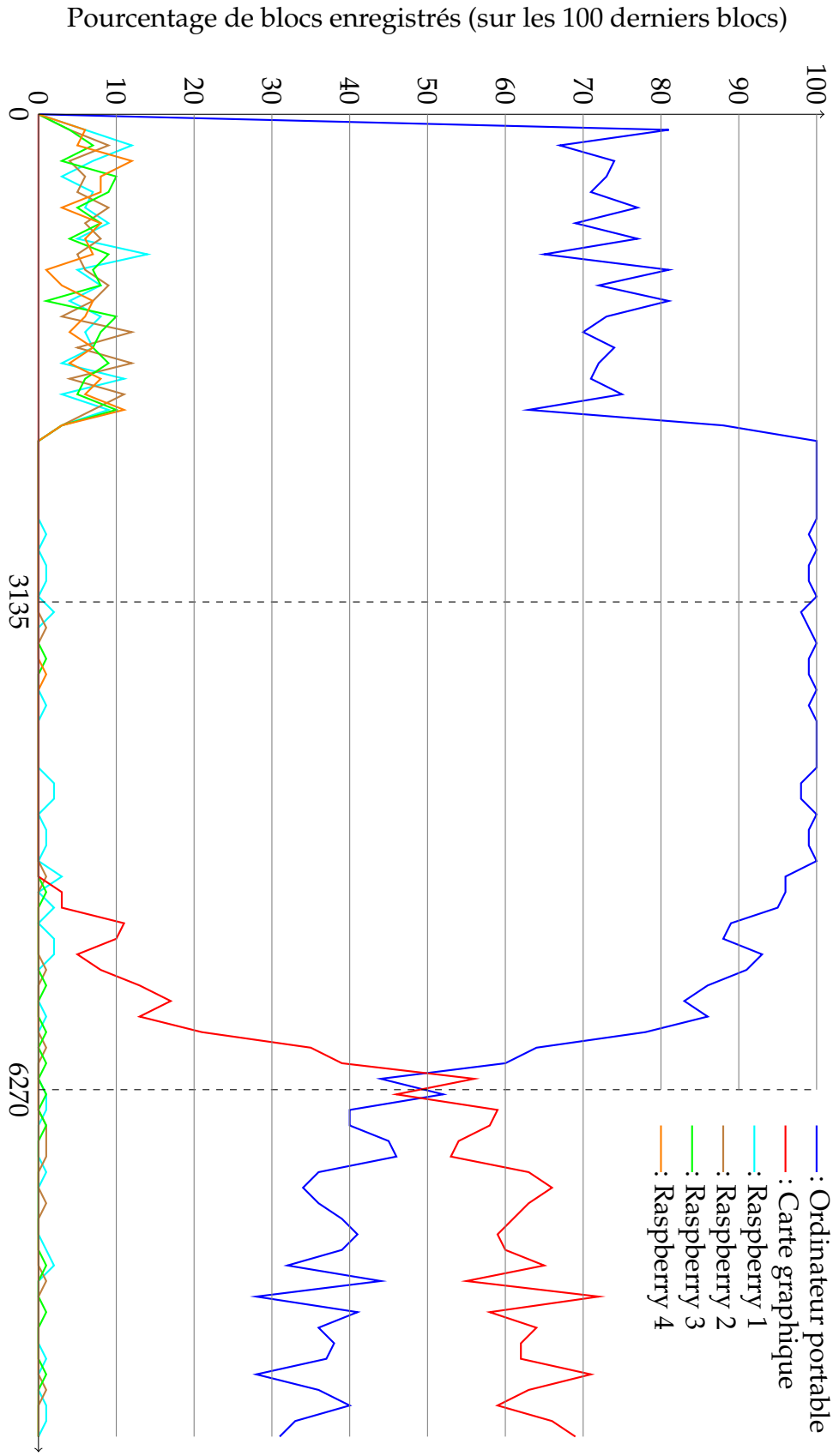


FIGURE 4.6 – *Nombre de blocs moyens enregistrés par chaque mineur au cours de la création de la chaîne (seconde fonction).*

d'inciter les mineurs du réseau à travailler sans discontinuité, ils ne peuvent pas tricher sans être pénalisés.

Nous avons défini les principaux critères que doivent proposer la fonction de calcul de la cible locale. De cette fonction dépend l'équilibre du réseau (notamment entre des mineurs avec peu de puissance mais beaucoup d'expérience et des mineurs avec une puissance importante et peu d'expérience), nous en avons proposé, implémenté et testé quelques unes.

La prochaine étape est de réfléchir à un système d'échange d'expérience. L'idée est de permettre aux mineurs du réseau de vendre leur expérience afin de la rentabiliser avant de quitter le réseau lorsqu'il le souhaite. ce système peut également permettre à un mineur de changer de compte (et donc de clef publique et privée) sans perdre son expérience.

Nos travaux ont été mis en valeur par une participation à une conférence internationale [48], une version étendue est en cours de rédaction.

V

Complexité pour le problème du jeu Ricochet Robots

À l'origine, nous voulions remplacer la preuve de travail utilisée par Bitcoin (le problème de nonce) par des instances du jeu de Ricochet Robots, toutefois, ce jeu s'est révélé être plus complexe que prévu et la génération d'instances aléatoires suivant une difficulté donnée est apparue inenvisageable. Malgré tout, nous avons analysé la complexité des problèmes correspondants au jeu de Ricochet Robots.

Ce chapitre présente nos travaux de complexité réalisés sur le jeu de société Ricochet Robots. Nous commençons par présenter le jeu ainsi que les résultats connus. Nous présentons quelques pré-requis afin de simplifier la compréhension des preuves.

Par la suite, nous présentons les trois résultats suivants : (1) la version optimisation de Ricochet Robots est **Poly-APX**-difficile, (2) la version décisionnelle est **PSPACE**-complète, (3) la version décisionnelle avec un nombre de robots infini et un plateau infini est indécidable. Ces résultats sont montrés par (1) une **S**-réduction depuis le problème de ENSEMBLE INDÉPENDANT MAXIMUM, (2) une réduction polynomiale depuis le TOKEN SLIDING et (3) une réduction polynomiale depuis toute machine de Turing.

5.1 Introduction

Ricochet Robots [8, 25, 27, 28] (voir figure 5.1) est un jeu de société créé par Alex Randolph édité en 1999. Le jeu se joue à deux ou plus, il est composé d'un plateau de jeu et de pions colorés appelés robots. Le plateau que nous appellerons grille de jeu est composé de cases. Certaines d'entre elles sont séparées par des murs placés sur l'arête les séparant. Un mouvement consiste à déplacer un robot verticalement ou horizontalement. Lorsqu'un mouvement est initié, le robot ne peut pas s'arrêter tant qu'il ne rencontre pas

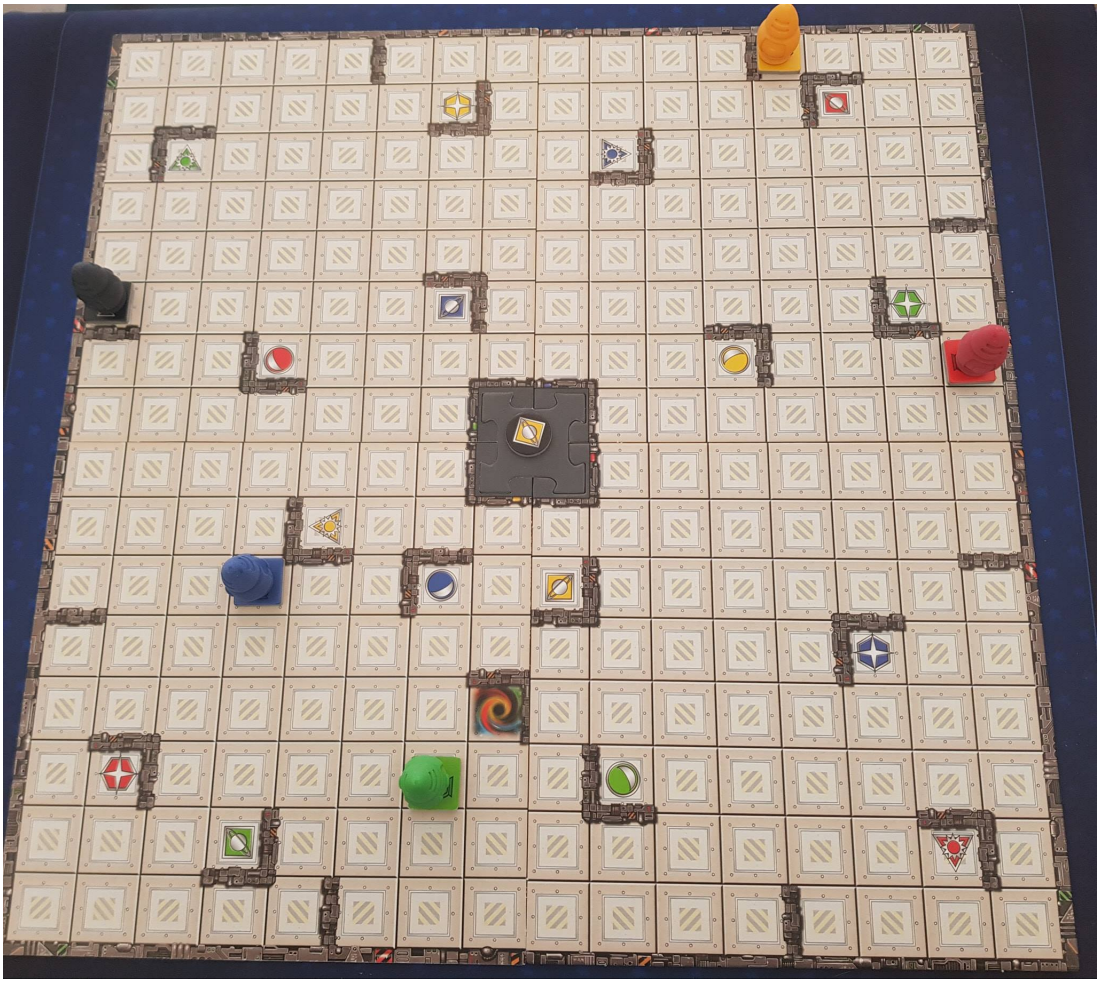


FIGURE 5.1 – *Photo du jeu Ricochet Robots. Dans cet exemple, le but est d'amener le robot jaune sur la case carré jaune (similaire au symbole au centre du plateau).*

d'obstacle (un mur ou un autre robot). Aucun mouvement ne peut être réalisé tant que le mouvement précédent n'est pas terminé.

Au départ, les robots sont disposés aléatoirement sur la grille de jeu. Le jeu se décompose en tour de jeu. À chaque tour, une case spéciale est tirée au sort (voir cases contenant des symboles dans la figure 5.1). Cette case, appelée case d'arrivée (ou case cible), est colorée par la même couleur que l'un des robots du jeu. Les joueurs ont alors trois minutes pour trouver comment amener le robot de même couleur sur cette case. Pour se faire, les joueurs ont le droit de déplacer tous les robots autant de fois qu'ils le désirent. Toutefois, le joueur qui arrive à déplacer le robot jusqu'à la case en un minimum de coups gagne un point. Les robots conservent le positionnement qu'ils ont à la fin de cette solution pour le tour suivant. Lorsque toutes les cases d'arrivée ont été jouées, la partie s'arrête. Le joueur ayant le plus de points gagne la partie. Dans le jeu original, le plateau de jeu est un carré de 16 cases de coté dans lequel les joueurs peuvent déplacer quatre robots. Beaucoup d'autres instances de Ricochet Robots peuvent être proposées avec des grilles de taille et de forme différentes, avec plus ou moins de robots, plus d'une case d'arrivée par tour, plusieurs robots d'une même couleur, pas de couleur... La figure 5.2 présente l'exemple d'une instance de Ricochet Robots sur une grille plus petite que la grille originale, avec deux robots colorés et deux cases d'arrivée colorées.

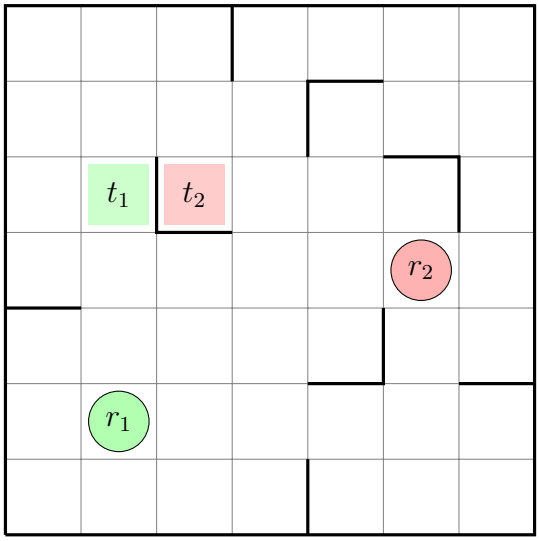


FIGURE 5.2 – *Exemple d'instance de Ricochet Robots composée de deux robots r_1 et r_2, ainsi que deux cases d'arrivée t_1 et t_2. Et on a r_1 et t_1 de la même couleur et r_2 et t_2 d'une autre couleur. Pour résoudre cette instance, il faut donc déplacer les robots pour atteindre une configuration telle que r_1 soit placé sur la case t_1 et que r_2 soit sur la case t_2. Remarque : étant donné que les cases d'arrivée ne bloquent pas les robots, r_1 aura nécessairement besoin de l'aide r_2 pour atteindre t_1 dans cette grille. Si l'on commence par placer r_2 avec les mouvements $\uparrow$ puis $\leftarrow$ et que l'on ne le déplace plus, r_1 ne pourra jamais atteindre t_1. Une solution possible pour résoudre cette instance est de réaliser les mouvements suivants : r_2: $\leftarrow$; r_1: $\uparrow, \leftarrow, \downarrow, \rightarrow$; r_2: $\uparrow, \rightarrow, \downarrow$.*

Ricochet Robots se place dans la catégorie des jeux de type « sliding games » comme le jeu de Push Push étudié par Demaine [19] ou encore le jeu Atomix, sorti en 1990 sur MS-DOS, étudié notamment par Holzer [35] et Huffner [37]. De nombreuses études ont été réalisées sur Ricochet Robots ou sur des jeux similaires. Icking [38, 39] considère le problème d'exploration d'une grille avec ou sans obstacles. Engels et Kamphans [25] étudient le problème d'existence d'une solution de Ricochet Robots avec n robots non colorés, une case d'arrivée et prouve que le problème est **PSPACE**-complet. Hesterberg [33] étudie la complexité de Ricochet Robots et Atomix paramétré par le nombre de robots (ou d'atomes pour Atomix) et la taille de la solution. Gebser [27, 28] propose d'utiliser le jeu Ricochet Robots comme benchmark pour des programmes ASP (answer set programming) et Butko [8] analyse les méthodes de raisonnements nécessaires pour résoudre des instances de Ricochet Robots.

5.2 Prérequis

5.2.1 Problèmes étudiés

Nous avons vu précédemment que le jeu original proposé par Alex Randolph implique à chaque tour quatre robots de couleurs différentes et une seule case d'arrivée (de couleur correspondante à un des robots). Engels et Kamphans [25] proposent un problème de décision correspondant au jeu de Ricochet Robots. Leur problème, appelé problème d'ACCESSIBILITÉ (reachability en anglais) est formulé de la façon suivante :

ACCESSIBILITÉ

Donnée : Considérant un plateau de jeu P de forme et de taille arbitraire, un système de n robots, $r_1, \ldots, r_n$, une configuration de départ $S = (s_1, \ldots, s_n)$ et une case d'arrivée t.

Question : Est ce qu'un des robots est capable d'atteindre la case t (possiblement avec l'aide d'autres robots) ?

Le problème proposé par Engels et Kamphans oublie de prendre en compte plusieurs paramètres. Pour commencer, le problème ne considère pas d'instance comprenant plus d'une case d'arrivée. De plus il ne prennent pas en compte les couleurs, que ce soit pour les robots ou pour la case d'arrivée. Nous avons donc choisi de redéfinir le problème afin de le rendre plus général et de prendre en compte les paramètres que nous venons d'énoncer.

Nous définissons une instance de Ricochet Robots comme un triplet $I = (P, R, T)$ constitué d'un plateau de jeu P, un ensemble de robots R et un ensemble de cases d'arrivée T. Les robots (et les cases d'arrivée) sont représentés par un couple (x, c) où x correspond à la case du plateau où se trouve le robot (ou la case d'arrivée) et c sa couleur. Une configuration (P, R, T) est qualifiée de gagnante si et seulement si toutes les cases d'arrivée contiennent un robot de même couleur que la case (i.e. si et seulement si $T \subseteq R$). Nous définissons le problème d'ACCESSIBILITÉ GÉNÉRALE (GRP pour General Reachability Problem) comme suit :

ACCESSIBILITÉ GÉNÉRALE

Donnée : Considérant une instance de Ricochet Robots $I = (P, R, T)$, avec P un plateau de jeu, R l'ensemble des robots et T l'ensemble des cases d'arrivée.

Question : Est-il possible de déplacer les robots (suivant la règle de déplacement de Ricochet Robots) afin d'atteindre une configuration gagnante ?

Le problème de l'ACCESSIBILITÉ GÉNÉRALE MAXIMUM (Max GRP pour Maximum General Reachability Problem) est le problème d'optimisation correspondant au problème d'ACCESSIBILITÉ GÉNÉRALE. La version d'optimisation ACCESSIBILITÉ GÉNÉRALE MAXIMUM est définie comme suit :

ACCESSIBILITÉ GÉNÉRALE MAXIMUM

Donnée : Considérant une instance de Ricochet Robots $I = (P, R, T)$, avec P un plateau de jeu, R l'ensemble des robots et T l'ensemble des cases d'arrivée.

Solution : Configuration accessible à partir de l'instance de départ après un ensemble de mouvements suivant les règles de déplacement de Ricochet Robots.

Mesure : $|K|$ avec K l'ensemble des cases d'arrivée atteintes $K \subseteq T$. Autrement dit, le nombre de cases d'arrivée sur lesquelles un robot de même couleur est positionné dans la configuration solution.

Holzer et al. [35] ont montré que le jeu Atomix est **PSPACE**-complet (à partir d'une réduction de problème d'intersection non-vide pour automates finis [45]). Engels et Kamphans [25] modifient cette preuve pour l'adapter au jeu de Ricochet Robots et montrent donc que le problème d'ACCESSIBILITÉ GÉNÉRALE est **PSPACE**-complet.

On peut facilement montrer que le problème d'ACCESSIBILITÉ GÉNÉRALE est polynomial pour tout nombre constant k de robots :

Théorème 3 : ACCESSIBILITÉ GÉNÉRALE admet un algorithme polynomial pour tout nombre constant k de robots (i.e. $|R| = k$, avec k constant).

Démonstration. Soit $I = (P, R, T)$ une instance du problème d'ACCESSIBILITÉ GÉNÉRALE avec un plateau de jeu P constitué de $|C|$ cases contenant k robots ($|R| = k$). Il existe $\binom{|C|}{k}$ configurations possibles (i.e. façons de disposer les robots sur le plateau). Considérons le graphe orienté G dans lequel chaque configuration possible correspond à un sommet et dans lequel il existe un arc de v_i vers v_j s'il est possible de passer en un seul mouvement de la configuration correspondante à v_i vers celle correspondante à v_j. S'il existe un chemin entre le sommet représentant la configuration de départ et un sommet représentant une configuration gagnante, alors I admet une solution. En utilisant un algorithme de parcours (tel que parcours en profondeur ou parcours en largeur), qui permet notamment de savoir en temps polynomial s'il existe un chemin entre deux points dans un graphe, nous pouvons déterminer en temps polynomial s'il existe ou non une suite de mouvements pour passer de la configuration de départ I à une configuration gagnante. Toutes les étapes décrites précédemment sont exécutées en temps polynomial, donc le problème d'ACCESSIBILITÉ GÉNÉRALE admet un algorithme polynomial pour tout nombre constant k de robots (peu importe le nombre de couleurs et le nombre de cases d'arrivée). ■

5.2.2 Représentation des instances générées

La plupart de nos preuves reposent sur des réductions d'un problème connu vers des instances du problème d'ACCESSIBILITÉ GÉNÉRALE (ou d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM). Nous proposons donc de construire des instances du problème d'ACCESSIBILITÉ GÉNÉRALE pour « simuler » d'autres problèmes dont nous connaissons la complexité. Les instances construites sont divisées en « morceaux » de grille, appelés gadgets, qui possèdent des propriétés structurelles spécifiques.

Afin d'améliorer la lisibilité de nos figures, les gadgets seront présentés séparément de nos constructions et seront représentés par des rectangles sur les figures. Les connexions entre gadgets seront faites par des « couloirs » (deux murs espacés d'une case l'une de l'autre) qui peuvent être droits ou pas et peuvent même se croiser (voir table 5.1). La possibilité de croiser deux connexions sans pour autant que le robot puisse changer de couloir permet de toujours respecter la planarité dans nos constructions même si deux connexions se croisent. En effet, un robot qui passe une intersection ne peut pas s'y arrêter et donc, si un robot rentre dans un croisement dans une direction verticale, il ne peut pas accéder à une sortie horizontale et vice versa.

Les connexions à sens unique (voir table 5.1) sont des connexions qui ne peuvent être traversées que dans un sens tant qu'il y a moins de robots que de cases bloquantes marquées par $\times$. Il est donc possible de bloquer k robots avec une connexion à sens unique de k cases bloquantes. À noter, dans l'exemple donné dans le table 5.1, si on considère k robots venant de la droite et un venant de la gauche, il est possible de placer $k-1$ robots sur les cases marquées d'un $\times$, de faire descendre le robot venant de la gauche, d'amener le dernier robot de droite vers celui-ci et faire passer les $k-1$ robots à contre-sens de la connexion. Dans ce cas là, $k-1$ robots ont traversé la connexion à contre-sens et l'augmentation du nombre de cases bloquantes n'aurait rien changé. Pour être certain d'éviter ce cas et empêcher tout robot de traverser à contre-sens il est possible d'enchaîner k connexions à sens unique.

Les connexions nécessitant une aide sont des connexions particulières. Prenons l'exemple donné dans le table 5.1, s'il y a un robot placé sur la case r (venant du couloir du haut) alors il peut donner la possibilité à un robot venant de la gauche d'atteindre la sortie du bas. Le robot ayant aidé l'autre peut ensuite ressortir par le haut ou horizontalement (nous verrons dans nos constructions que nous ne ferons jamais sortir le robot horizontalement, s'il le fait, il se retrouvera forcément bloqué). Si aucun robot n'est présent pour aider, le robot venant de la gauche ne s'arrête pas et ressort par la droite.

Dans nos preuves, considérant r_1 le robot aidant et r_2 le robot nécessitant de l'aide, on dit que « r_1 intercepte r_2 pour atteindre la sortie correcte (i.e. celle du bas dans le table 5.1) » lorsque r_1 se place sur la case r pour aider r_2 à atteindre la sortie du bas. De même, considérant deux gadgets G_1 et G_2, « G_1 intersecte le couloir c vers G_2 » signifie qu'une sortie de G_1 et une entrée de G_2 sont connectées au couloir c de façon à ce qu'un robot venant de G_1 peut se placer pour aider un autre robot à atteindre le gadget G_2.

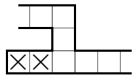
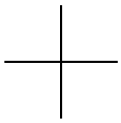
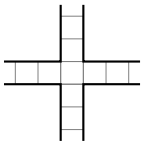
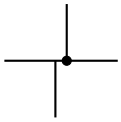
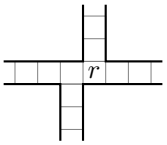
	Représentation	Grille correspondante
Connexion		
Virage dans une connexion		
Connexion à sens unique		
Croisement		
Connexion nécessitant une aide		

TABLE 5.1 – *Représentation des connexions entre les gadgets.*

5.2.3 Gadgets de base et propriétés structurelles

Certains des gadgets que nous avons définis sont utilisés pour plusieurs preuves. Nous présentons ces gadgets et leurs propriétés dans dans cette sous-section. Ces gadgets seront ensuite réutilisés dans des constructions plus complexes.

Un k-*routeur* (ou *routeur*) présenté dans la figure 5.3 est un gadget possédant une seule entrée et k sorties (avec $k \in \mathbb{N}^*$). Le nombre de sorties k est appelé taille du *routeur*. Le robot peut toujours faire le tour du gadget (dans le sens anti-horaire) et atteindre la sortie qu'il souhaite. Les gadgets k-*routeur* possèdent la propriété suivante :

Propriété 3 : [Propriété de routeur] Lorsqu'un robot accède à un k-routeur (par le biais d'une entrée ou d'une sortie), il peut accéder à n'importe quelle sortie.

Un k-*dispatcheur* (voir figure 5.5) est un gadget possédant une seule entrée et k sorties (avec $k \in \mathbb{N}^*$) comme le k-router. Toutefois, contrairement à un k-router, lorsqu'un robot sort d'un k-dispatcheur par une sortie s, il ne peut pas revenir sur ses pas pour accéder à une autre sortie s'.

Un k-*synchroniseur* est un gadget avec k entrées et k sorties (avec $k \in \mathbb{N}$, $k > 1$). La figure 5.6 et la figure 5.7 représentent un *2-synchroniseur* et un *3-synchroniseur*. Les gadgets k-*synchroniseur* possèdent tous la propriété suivante :

Propriété 4 : [Propriété de synchroniseur] Soit un nombre de robots k' qui accèdent à un gadget k-*synchroniseur* (noté G). Si $k' < k$, alors aucun robot ne peut sortir du gadget. Si $k' = k$, alors tous les robots peuvent atteindre une sortie, toutefois, deux robots ne peuvent pas atteindre la même sortie.

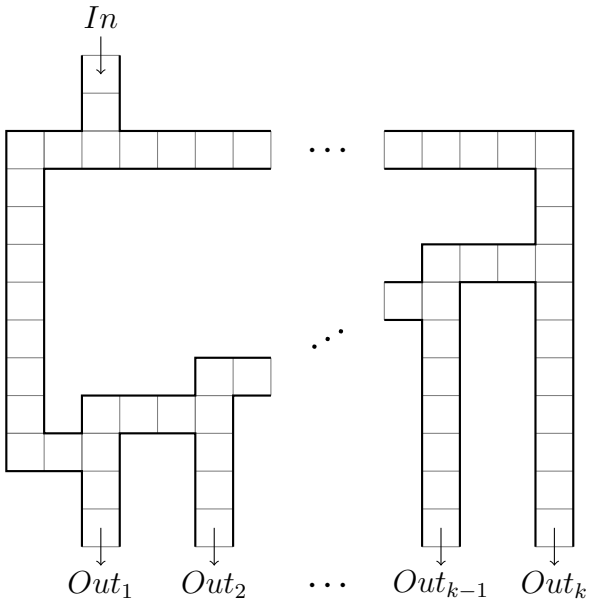


FIGURE 5.3 – *Gadget* k-routeur. *Ce gadget possède une entrée et k sorties. Dans ce gadget, un robot peut accéder à la sortie qu'il désire (en se déplaçant dans le sens anti-horaire dans le gadget).*

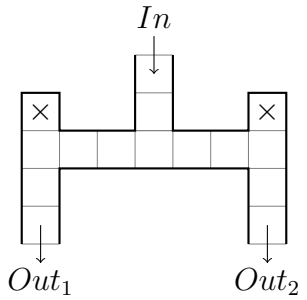


FIGURE 5.4 – *Gadget 2-dispatcheur, les cases marqués par $\times$ correspondent aux cases qui permettent de bloquer un robot s'il essaye de passer d'une sortie à une autre.*

Introduisons la propriété de k-non-retour (pour $k \in \mathbb{N}^*$) :

Propriété 5 : [Propriété de k-non-retour] Si k robots ou moins accèdent à un gadget possédant une propriété de k-non-retour et que chacun d'entre eux a réalisé un mouvement dans le gadget, alors aucun d'entre eux ne peut sortir du gadget par une entrée du gadget.

À noter qu'un gadget possédant une propriété de k-non-retour possède également une propriété de $(k-1)$-non-retour $\forall k \geq 1$.

L'ensemble des gadgets que nous définissons possèdent une propriété de 1-non-retour (dont les gadgets de types router et dispatcheur). Les gadgets k-synchroniseur possèdent une propriété de k-non-retour.

Dans l'ensemble de nos constructions, les gadgets sont connectés de façon à ce qu'il n'y ait jamais plus d'un robot dans les gadgets de types router, dispatcheur et jamais plus de k robots dans un gadget du type k-synchroniseur. De plus, si la propriété de non-retour ne suffisait pas à assurer que des robots puissent passer de l'entrée d'un gadget vers la sortie

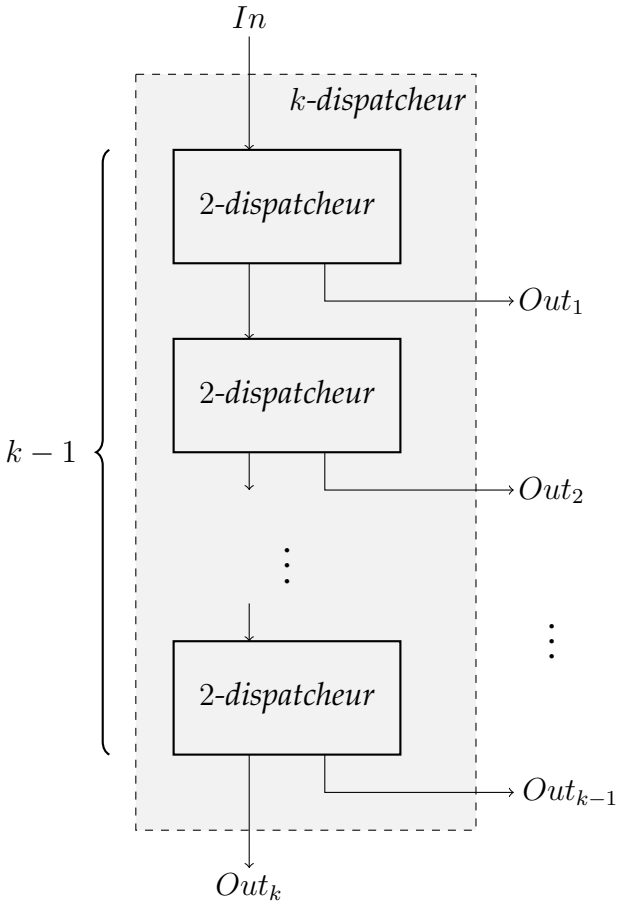


FIGURE 5.5 – *Gadget k-dispatcheur. Construction d'un gadget k-dispatcheur à partir de $k-1$ gadgets 2-dispatcheur.*

du précédent, il est possible d'ajouter des connexions à sens unique de taille nécessaire.

5.3 Complexité du problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM

Dans cette section, nous présentons une **S**-réduction du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM vers le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM. Cette réduction, qui préserve le ratio d'approximation du problème d'origine, nous permet de montrer notamment que le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM appartient à la classe d'approximation **Poly-APX**-difficile.

Nous rappelons la définition de la **S**-réduction dans la section 1.4.3. Le problème de l'ENSEMBLE INDÉPENDANT MAXIMUM est défini comme suit :

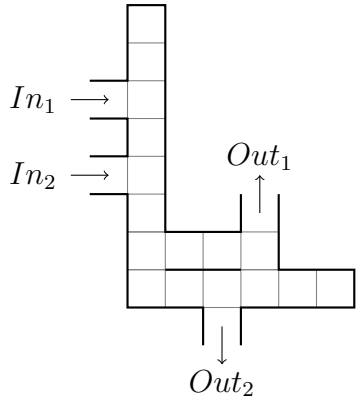


FIGURE 5.6 – *Gadget* 2-synchroniseur*, deux entrées et deux sorties. Considérons deux robots r_1 et r_2 qui rentrent dans le gadget par les entrées In_1 et In_2 respectivement. La seule solution que ces robots ont pour sortir du gadget 2-synchroniseur est de collaborer afin que l'un puisse atteindre la sortie Out_1 et l'autre la sortie Out_2. Notez que les deux robots peuvent atteindre les deux sorties mais lorsqu'un robot choisit une sortie, l'autre n'a pas le choix que de prendre l'autre. Mouvements possibles pour traverser un gadget 2-synchroniseur à partir du moment où r_1 et r_2 ont atteint le gadget par les entrées In_1 et In_2 respectivement : r_2: $\downarrow$; r_1: $\downarrow, \rightarrow, \downarrow$; r_2: $\rightarrow, \downarrow$; r_1: $\uparrow$.*

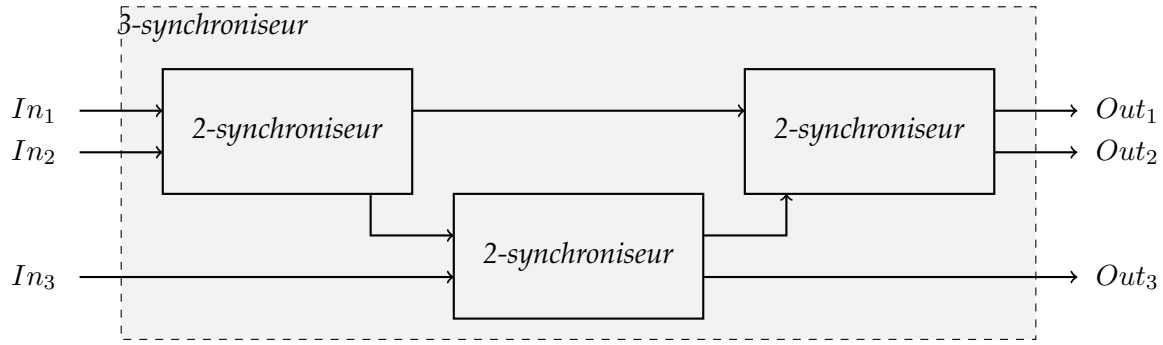


FIGURE 5.7 – *Gadget 3-synchroniseur. En réutilisant la structure de ce gadget, il est possible de généraliser la construction d'un gadget k-synchroniseur en remplaçant les gadgets 3-synchroniseur par des gadgets $(k-1)$-synchroniseur.*

ENSEMBLE INDÉPENDANT MAXIMUM

Donnée : Soit $G = (V, E)$ un graphe.
Solution : Un ensemble indépendant de sommets, i.e. un sous-ensemble $V' \subseteq V$ tel que aucune paire de sommets de V' ne soit relié par une arrête de E.
Mesure : Cardinalité de l'ensemble indépendant, i.e. $|V'|$

Le problème de l'ENSEMBLE INDÉPENDANT MAXIMUM à été prouvé **Poly-APX**-complet par Bazgan et al. [3].

Dans notre preuve, nous considérons la fonction $N(v_i)$ qui retourne l'ensemble des sommets voisins du sommet v_i. Pour que notre **S**-réduction soit correcte, il faut imaginer une construction telle que l'ensemble des sommets indépendants soit de taille strictement égal au nombre de cases d'arrivée recouvertes par un robot.

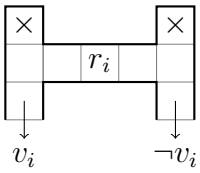


FIGURE 5.8 – *Gadget v_i-CHOIX. La case marquée par r_i correspond à la position de départ du robot r_i. Si un robot essaie de rentrer dans le gadget par une sortie, il est bloqué par les cases marquées par $\times$ et ne pourra pas atteindre l'autre sortie.*

L'idée principale de notre construction est de représenter chaque sommet de V par un robot. Il y aura donc $|V|$ robots dans l'instance générée, chacun d'entre eux ne pourra espérer atteindre qu'une seule des $|V|$ cases d'arrivée, et ce, seulement s'il obtient l'aide de l'ensemble des robots représentant ses voisins dans le graphe correspondant. On aura donc une instance telle que $|R| = |T| = |V|$. De plus, dans notre construction, un robot qui aide un de ses voisins à atteindre sa case d'arrivée ne pourra jamais atteindre la sienne.

Dans la construction suivante, nous ne parlerons pas de couleurs de robots ni de couleurs de cases car les instances générées seront des instances monochromes (tous les robots et toutes les cases d'arrivée partagent une seule et même couleur).

Pour notre construction, nous aurons besoin de trois gadgets différents, un exemplaire de chacun d'entre eux sera généré pour chaque robot de l'instance (et donc pour chaque sommet de V).

La première catégorie de gadgets que nous utilisons sont des 2-dispatcheurs appelés v_i-CHOIX (voir figure 5.8). Ces gadgets, dont leur nombre est égal au nombre de robots de notre construction, représentent le choix que fait chaque robot entre aider ses voisins à atteindre leur case d'arrivée respective ou essayer d'atteindre sa propre case d'arrivée. À noter que, étant donné qu'il s'agit de gadgets de type k-dispatcheur, lorsqu'un robot accède à une sortie du gadget, il ne peut pas revenir pour en choisir une autre, i.e. toute sortie est définitive.

La seconde catégorie de gadgets que nous utilisons sont des gadgets que nous appelons v_i-SOMMET (voir figure 5.9). Ces gadgets permettent de vérifier l'absence de l'ensemble des voisins de v_i dans la solution finale quand nous essayons d'ajouter v_i à la solution. Dans ce gadget, seul le robot en provenance de l'entrée v_i peut atteindre la case d'arrivée t_i si et seulement si les $|N(v_i)|$ autres entrées sont atteintes par un robot (autrement le robot sera bloqué dans les cases marquées par des $\times$). À noter qu'un robot qui accède à une entrée $e_{v_i,\neg v_j}$ peut aider le robot venant de l'entrée v_i avant de ressortir par le couloir d'où il est arrivé. Toutefois, un robot rentrant dans un gadget de type v_i-SOMMET ne peut pas en sortir par un autre couloir que celui d'où il vient (i.e. chaque étage du gadget possède la propriété de 2-non retour, et nous verrons avec les connexions qu'il ne peut pas y avoir plus de deux robots dans l'étage d'un gadget de type v_i-SOMMET).

Lemme 1 : Un robot r_i rentrant dans un gadget v_i-SOMMET par l'entrée v_i peut atteindre la case d'arrivée t_i si et seulement si $\forall v_j \in N(v_i)$, l'entrée $e_{v_i,\neg v_j}$ est atteinte par un robot.

Démonstration. Si $\forall v_j \in N(v_i)$, l'entrée $e_{v_i,\neg v_j}$ est atteinte par un robot, alors r_i peut

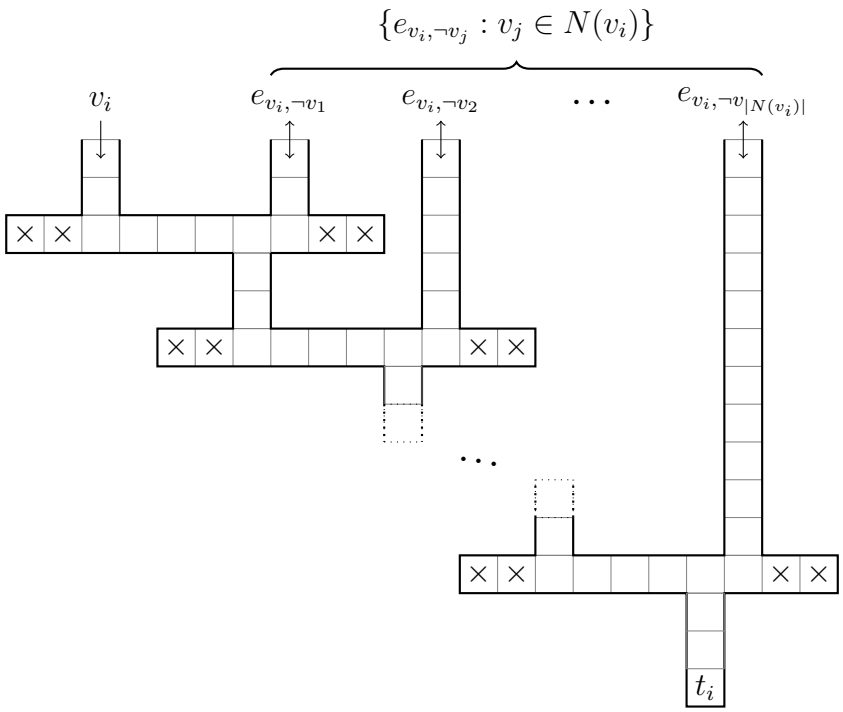


FIGURE 5.9 – *Gadget* v_i-SOMMET. *Considérant un sommet* $v_i \in V$*, le gadget* v_i-SOMMET *correspondant est constitué de* $|N(v_i)|+1$ *entrées. Seul le robot venant de l'entrée* v_i *peut atteindre la case d'arrivée* t_i*. Pour l'atteindre, il a besoin de l'aide successive de robots venant des entrées de type* $e_{v_i,\neg v_j}$ *(*$\forall v_j \in N(v_i)$*), autrement le robot venant de l'entrée* v_i *ne sera pas en mesure d'atteindre la case* t_i *et sera bloqué dans les cases marquées par* $\times$*.*

s'appuyer sur eux les uns après les autres pour accéder à la case t_i. Considérons qu'aucun robot n'atteigne l'entrée $e_{v_i,\neg v_j}$, alors r_i sera bloqué dans les cases marquées par $\times$ dans l'étage correspondant. Dans ce cas, r_i ne sera pas en capacité d'atteindre la case t_i. ■

La dernière catégorie de gadgets dont nous avons besoin sont des gadgets de type routeur. Ces gadgets, que nous appelons $\neg v_i$-ROUTEUR, permettent aux robots d'aider leurs voisins à atteindre leur case d'arrivée respective. Les sorties d'un gadget$\neg v_i$-ROUTEUR sont reliées à une entrée de chaque gadget v_j-SOMMET tel que $v_j \in N(v_i)$).

Rappelons que, d'après la propriété 3, un robot qui accède à un gadget de type routeur (que ce soit par le biais d'une entrée ou d'une sortie) peut toujours atteindre n'importe quelle sortie (en circulant dans le gadget dans le sens anti-horaire).

Remarque 1. Dans les instances que nous construisons, chaque sommet $v_i \in V$ est représenté par un robot $r_i \in R$. Chaque robot r_i ne peut accéder qu'a une seule et unique case d'arrivée t_i. Chaque robot à le choix entre essayer d'atteindre sa case d'arrivée et aider ses voisins à atteindre les leurs. Autrement dit, un robot qui aide un (ou plusieurs) voisin ne peut plus prétendre accéder à sa case d'arrivée. L'idée principale de l'instance que nous construisons repose sur le fait qu'un robot r_i ne peut accéder à sa case d'arrivée t_i que s'il peut traverser le gadget v_i-SOMMET et donc seulement si tous ses voisins l'aident. Donc si un robot r_i accède à sa case d'arrivée t_i, l'ensemble de ses voisins ne peuvent plus faire partie de la solution.

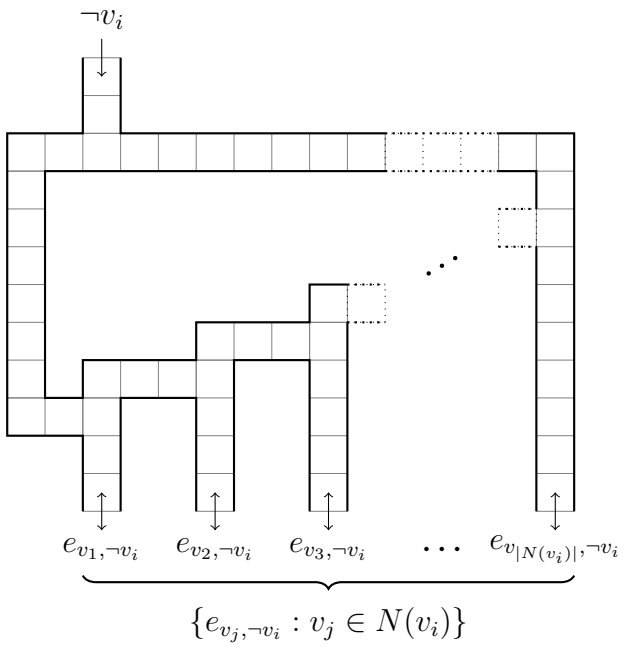


FIGURE 5.10 – *Gadget $\neg v_i$-*ROUTEUR. *Considérant un sommet $v_i \in V$, le gadget $\neg v_i$-*ROUTEUR *correspondant est constitué de $|N(v_i)| - 1$ sorties, qu'il peut également utiliser en entrée. Un robot entrant dans un $\neg v_i$-*ROUTEUR *peut toujours atteindre la sortie qu'il souhaite, peu importe d'où il vient. Toutefois il ne peut pas accéder à $\neg v_i$ (il s'agit d'une entrée).*

Construction 1 : Soit I une instance de ENSEMBLE INDÉPENDANT MAXIMUM, un graphe $G = (V, E)$ et la fonction $N(v_i)$ qui retourne l'ensemble des voisins du sommet v_i ($\forall v_i \in V$). L'instance $I' = (P, R, T)$ du problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM est construite de la manière suivante :

- $\forall v_i \in V$ nous considérons un robot $r_i \in R$ ainsi qu'une case d'arrivée $t_i \in T$.
- $\forall r_i \in R$ nous construisons un gadget v_i-CHOIX (i.e. un 2-dispatcheur), un gadget $\neg v_i$-ROUTEUR de $|N(v_i)|$ sorties (i.e. un $|N(v_i)|$-routeur) et un gadget v_i-SOMMET de $|N(v_i)|$ entrées de type $e_{v_i,\neg v_j}$ (voir figure 5.9).
- $\forall r_i \in R$, r_i commence dans le gadget v_i-CHOIX correspondant.
- $\forall v_i \in V$, la sortie v_i du gadget v_i-CHOIX est connectée à l'entrée v_i du gadget v_i-SOMMET alors que la sortie $\neg v_i$ est connectée à l'entrée $\neg v_i$ du gadget $\neg v_i$-ROUTEUR.
- Chaque sortie $e_{v_i,\neg v_j}$ du gadget $\neg v_j$-ROUTEUR est connectée à l'entrée $e_{v_i,\neg v_j}$ du gadget v_i-SOMMET. Afin de donner une intuition au lecteur,pour chaque arête $(v_i, v_j) \in E$ nous connectons la sortie $e_{v_i,\neg v_j}$ du gadget $\neg v_j$-ROUTEUR avec l'entrée $e_{v_i,\neg v_j}$ du gadget v_i-SOMMET ainsi que la sortie $e_{v_j,\neg v_i}$ du gadget $\neg v_i$-ROUTEUR avec l'entrée $e_{v_j,\neg v_i}$ du gadget v_j-SOMMET.
- Chaque case d'arrivée $t_i \in T$ est placée dans le gadget v_i-SOMMET correspondant.

Clairement, la transformation décrite précédemment s'exécute en temps polynomial par rapport à la taille du graphe (i.e. $|V|$). La figure 5.11 présente un exemple d'instance I du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM et l'instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM obtenue par construction 1.

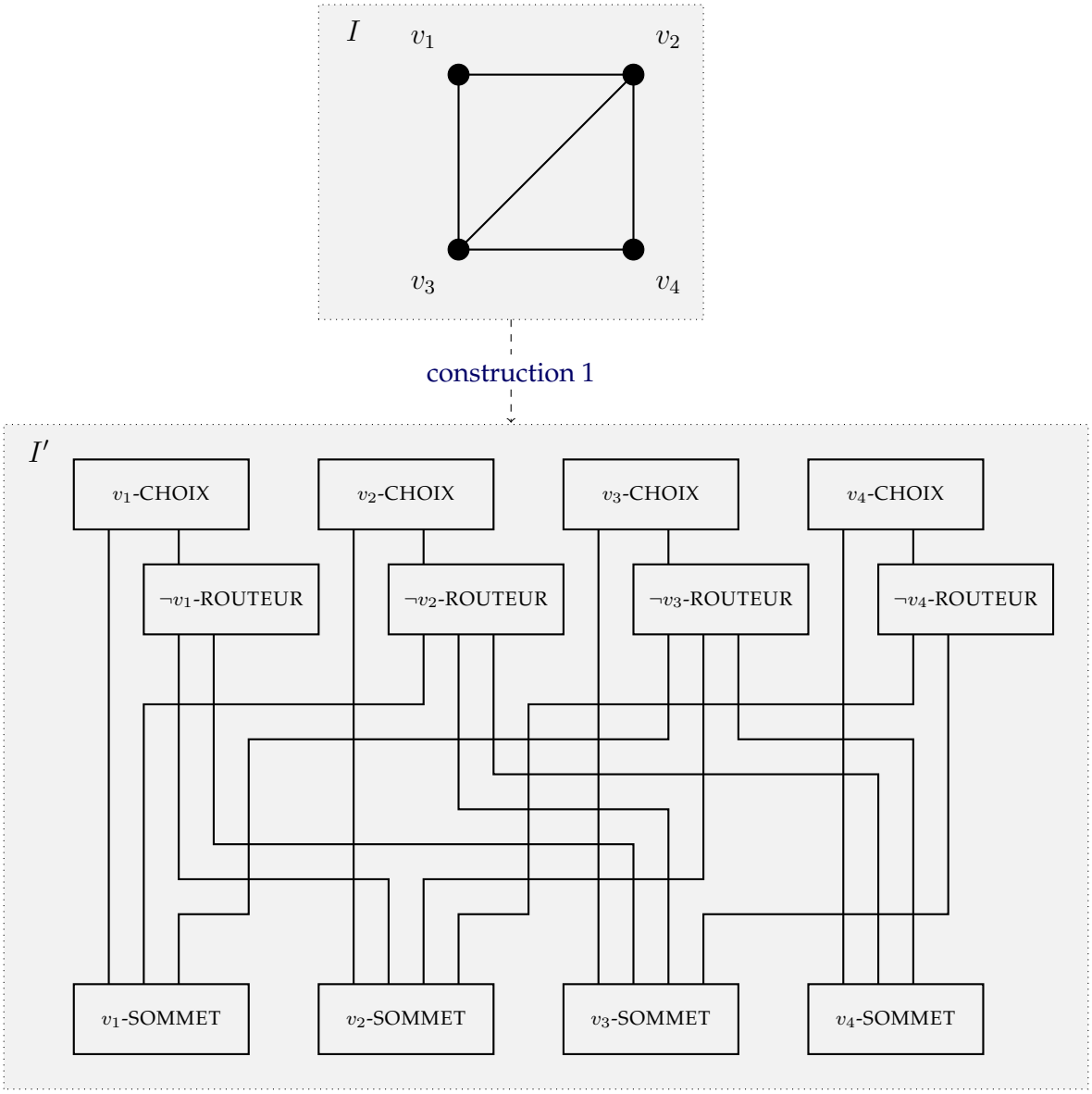


FIGURE 5.11 – *Exemple d'instance I' du problème d'*ACCESSIBILITÉ GÉNÉRALE MAXIMUM *obtenue suivant la construction 1 à partir de l'instance I du problème de l'*ENSEMBLE INDÉPENDANT MAXIMUM.

Théorème 4 : Il existe une réduction polynomiale du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM vers le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM.

Démonstration. Soit I une instance du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM et I' l'instance du problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM obtenue par la construction 1, I admet une solution maximale de taille $|V'| = k$ si et seulement si I' admet une solution maximale de taille $|K| = k$ (i.e. il est possible d'atteindre k cases d'arrivées

avec au plus k robots).

- Supposons que I admet une solution maximale de taille $|V'| = k$, alors I' admet une solution maximale de taille k :

  Soit $V' \subseteq V$ une solution maximale de I. Supposons que $\forall v_i \in V$, si $v_i \in V'$ alors le robot correspondant (r_i) sort du gadget v_i-CHOIX par la sortie v_i et accède au gadget v_i-SOMMET sinon r_i prend la sortie $\neg v_i$ et accède au gadget $\neg v_i$-ROUTEUR. En accord avec le lemme 1, r_i peut atteindre la case d'arrivée t_i si et seulement si toutes les entrées du gadget v_i-SOMMET sont atteintes par un robot (autrement dit, si $\forall v_j \in N(v_i)$, le robot correspondant r_j a atteint le gadget $\neg v_j$-ROUTEUR). Ainsi, r_i peut accéder à la case d'arrivée t_i et, d'après la propriété de non-retour des gadgets de type routeur (propriété 5), aucun des robots r_j ($\forall v_j \in N(v_i)$) ne peut atteindre une case d'arrivée. Supposons qu'il existe un robot $r_l \in R$ pouvant atteindre une case d'arrivée tel que $v_l \notin V'$. Alors $\exists v_l \in V$ tel qu'aucun de ses voisins n'appartienne à V' et $v_l \notin V'$. L'ensemble $\{v_l\} \cup V'$ constitue une solution de l'instance I de taille $k+1$, V' n'est pas une solution maximale, on arrive sur une contradiction, donc $\nexists r_l \in R$ pouvant atteindre une case d'arrivée tel que $v_l \notin V'$. Conclusion, si I admet une solution maximale de taille k alors I' admet également une solution maximale de taille k.

- Réciproquement, supposons que I' admet une solution maximale de taille $|K| = k$, alors I admet une solution de taille k :

  Si I' admet une solution maximale de taille k, alors k robots ont atteint une case d'arrivée. En accord avec lemme 1, r_i peut atteindre la case d'arrivée t_i du gadget v_i-SOMMET si et seulement si $\forall v_j \in N(v_i)$, l'entrée $e_{v_i,\neg v_j}$ est atteinte par un autre robot. D'après les connexions décrites dans la construction 1, $\forall v_j \in N(v_i)$, un robot peut accéder à l'entrée $e_{v_i,\neg v_j}$ seulement s'il provient du gadget $\neg v_j$-ROUTEUR et donc seulement si r_j a choisi la sortie $\neg v_j$ du gadget v_j-CHOIX. Ainsi, les k cases d'arrivées atteintes par des robots correspondent à un ensemble V' de k sommets tel qu'aucun de leurs voisins appartient à l'ensemble (i.e. un ensemble indépendant de taille k). Supposons qu'il existe un sommet $v_l \in V \backslash V'$ tel que $\{v_l\} \cup V'$ est un ensemble indépendant. Dans ce cas, il existe une case d'arrivée $t_l \in T \backslash K$ qui peut être atteinte par un robot r_l. En effet, puisque $\forall v_j \in N(v_l)$, $v_j \notin V'$ alors $\forall v_j \in N(v_l)$ la case d'arrivée correspondante $t_j \notin K$ et donc le robot correspondant r_j accède au gadget $\neg v_j$-ROUTEUR sans changer la solution. Donc le robot r_l peut accéder au gadget v_l-SOMMET et atteindre la case d'arrivée t_l. Dans ce cas, $\{t_l\} \cup K$ constitue une solution de l'instance I' de taille $k+1$, K n'est pas une solution maximale, on arrive sur une contradiction, donc $\nexists v_l \in V \backslash V'$ tel que $\{v_l\} \cup V'$ est un ensemble indépendant. Conclusion, si I' admet une solution maximale de taille k alors I admet également une solution maximale de taille k.

Étant donné que toute solution maximum est une solution maximale, alors une instance I du problème de Max IS admet une solution maximum de taille $|V'| = k$ si et seulement si l'instance I' de Max GRP obtenue par la construction 1 admet une solution maximum de taille $|K| = k$.

Par les arguments précédemment énoncés, il existe une réduction polynomiale du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM vers le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM. ■

Théorème 5 : Le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM est **Poly-APX**-difficile.

Démonstration. Pour prouver ce théorème, nous montrons qu'il existe une **S**-réduction du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM vers le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM. Soit Π le problème de l'ENSEMBLE INDÉPENDANT MAXIMUM, Π' le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM, I une instance de Π, I' une instance de Π', V' une solution de I et K une solution de I'. Nous avons :

- Une fonction $f(I)$ qui construit I' en temps polynomial (construction 1).
- Une fonction $g(K)$ qui construit V' en temps polynomial ($V' = \{v_i : \ t_i \in K\}$).

Et :

1. Pour toute instance I de Π, $OPT_{\Pi}(I) = OPT_{\Pi'}(f(I))$ (théorème 4).
2. Pour toute instance I de Π et pour toute solution réalisable K d'une instance $f(I)$ de Π', $m_{\Pi}(I, g(K)) = m_{\Pi'}(f(I), K)$ (théorème 4).

Étant donné que la réduction décrite ci-dessus est une **S**-réduction du problème de l'ENSEMBLE INDÉPENDANT MAXIMUM vers le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM et que le problème de l'ENSEMBLE INDÉPENDANT MAXIMUM est **Poly-APX**-complet (Bazgan et al. [3]), cette réduction prouve que le problème d'ACCESSIBILITÉ GÉNÉRALE MAXIMUM est **Poly-APX**-difficile. ■

5.4 Complexité du problème d'ACCESSIBILITÉ GÉNÉRALE

Engels et Kamphans [26] ont montré que le problème d'ACCESSIBILITÉ GÉNÉRALE est **PSPACE**-complet en modifiant la preuve de Holzer et al. [35] montrant que le jeu Atomix est **PSPACE**-complet (à partir d'une réduction de problème d'intersection non-vide pour automates finis [45]). Dans cette section nous proposons une preuve alternative à celle proposée par Engels et Kamphans. Nous proposons une réduction polynomiale du problème de reconfiguration d'ensemble indépendant (TOKEN SLIDING) vers le problème d'ACCESSIBILITÉ GÉNÉRALE. Étant donné que le problème de TOKEN SLIDING a été montré **PSPACE**-complet [40], cette réduction réprouve l'appartenance du problème d'ACCESSIBILITÉ GÉNÉRALE à la classe **PSPACE**-difficile. Nous montrons également que le problème appartient à la classe **PSPACE** afin de conclure sur son appartenance à la classe **PSPACE**-complet.

Kamiński et al. [43] définissent les problèmes de reconfiguration comme suit : étant donné deux solutions réalisables x, y de l'instance I, l'objectif est de trouver une séquence de reconfigurations $s_1, \ldots, s_k$ tel que $s_1 = x$, $s_k = y$, et tel que tout s_i (pour $1 < i < k$) est

une solution réalisable de I, et telles que les transitions entre s_i et s_{i+1} soient permises par les règles de reconfiguration.

Le problème de TOKEN SLIDING (ou problème de reconfiguration d'ensembles indépendants) est un problème de reconfiguration de graphe. Les jetons (i.e. les tokens) ne peuvent être placés que sur des sommets du graphe et il est possible de déplacer un jeton d'un sommet à un autre seulement si les deux sommets sont voisins et aucun jeton ne se trouve sur un des voisins du sommet d'arrivée. Étant donné un graphe et deux dispositions de jetons sur le graphe, est-il possible de déplacer les jetons depuis une configuration pour obtenir la seconde ? Plus formellement, le problème de TOKEN SLIDING peut être défini comme suit :

TOKEN SLIDING

Donnée : Un graphe $G = (V, E)$ et deux ensembles indépendants A et B de G.
Question : Est-il possible de reconfigurer A en B via une séquence S d'ensembles indépendants (numérotés $s_1, \ldots, s_{|S|}$) tel que chacun d'entre eux résulte du précédent auquel on enlève un sommet et ajoute un de ses voisins.

Une opération de sliding (ou sliding) consiste à modifier un ensemble de sommets S en lui retirant un sommet v_i ($\in S$) et en ajoutant un sommet v_j tel que v_j et v_i sont voisins ($(v_i, v_j) \in E$). Soit S' l'ensemble obtenu par une opération de sliding sur S, on a $S' = \{v_j\} \cup S \backslash \{v_i\}$. Considérant le problème de TOKEN SLIDING, une opération de sliding est dite valide (ou légale) si l'ensemble S' obtenu après l'opération est un ensemble indépendant.

Dans cette preuve, nous considérons $n = |V|$, $k = |A| = |B|$ et la fonction $N(v_i)$ qui retourne l'ensemble des voisins du sommet v_i. Nous définissons des gadgets afin de transformer un ensemble indépendant s_i en un autre ensemble s_{i+1} obtenu en retirant un sommet v_j de s_i et en ajoutant un sommet v_k parmi ses voisins ($v_k \in N(v_j)$). Nous réutilisons la construction et les gadgets de la preuve précédente (voir section 1.4.3) afin de montrer à chaque étape que l'ensemble obtenu (s_{i+1}) est un ensemble indépendant.

Tout comme la preuve précédente, nous commençons par présenter les gadgets et propriétés nécessaires à la construction finale.

Les deux premiers gadgets utilisés sont de type routeur. Ces gadgets, que nous appelons v_i-ROUTEUR (voir figure 5.12) et $\neg v_i$-ROUTEUR (voir figure 5.13), respectent la propriété 3 (un robot dans ces gadgets peut accéder à n'importe quelle sortie, peut importe d'où il vient). Les gadgets v_i-ROUTEUR et $\neg v_i$-ROUTEUR possèdent respectivement $|V| + 1$ et $3|V|$ sorties.

Le troisième gadget, que nous appelons v_i-DISPATCHEUR, est de type dispatcheur. Pour chaque sommet $v_i \in V$, le gadget v_i-DISPATCHEUR correspondant possède $|N(v_i)|$ sorties. Voir figure 5.14 pour plus de détails sur le nom des entrées et des sorties que nous utilisons.

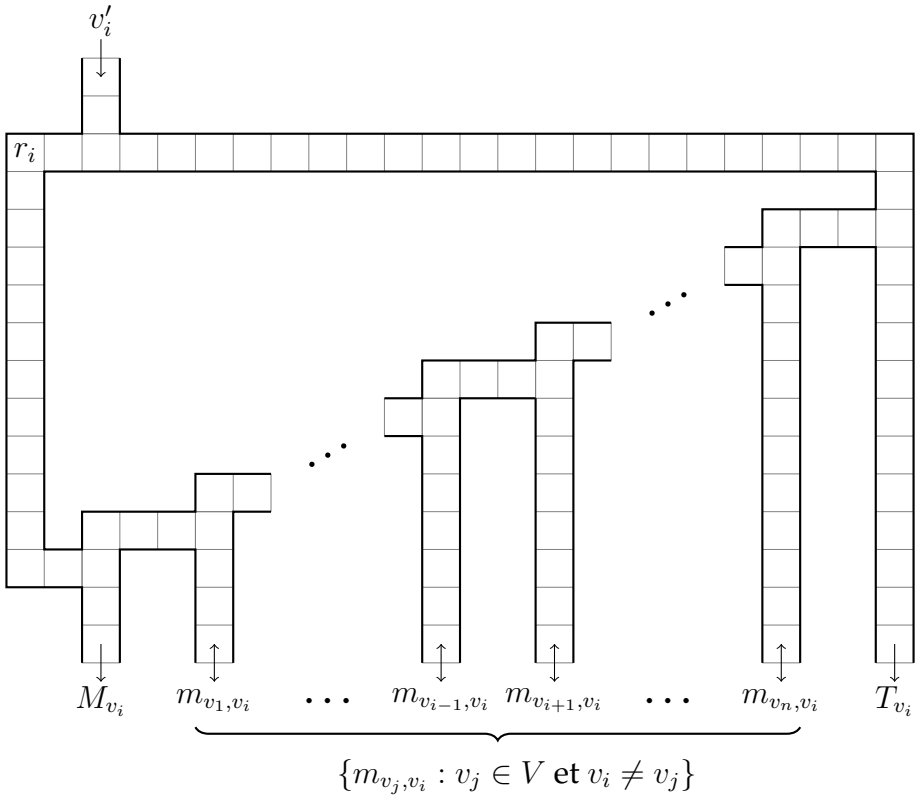


FIGURE 5.12 – *Gadget* v_i-ROUTEUR. *Un gadget* v_i-ROUTEUR *possède* $|V|+1$ *sorties, qu'il peut également utiliser en entrée. Un robot entrant dans un* v_i-ROUTEUR *peut toujours atteindre la sortie qu'il souhaite, peu importe d'où il vient. Toutefois il ne peut pas accéder à* v'_i *(il s'agit d'une entrée). Le robot* r_i *a pour position de départ la case marquée* r_i *si et seulement si le sommet correspondant* v_i *appartient à l'ensemble indépendant de départ* A *(si et seulement si* $v_i \in A$*).*

Le gadget suivant, appelé v_i-SOMMET (voir figure 5.15), correspond au gadget du même nom de la preuve précédente (figure 5.9) obtenu en remplaçant la case d'arrivée par une sortie v'_i. Ce nouveau gadget conserve ainsi la propriété du gadget figure 5.9 de la preuve précédente :

Propriété 6 : [Propriété de v_i-SOMMET] Un robot r_i rentrant dans un gadget v_i-SOMMET par l'entrée v_i peut atteindre la sortie v'_i si et seulement si $\forall v_j \in N(v_i)$, l'entrée $e_{v_i,\neg v_j}$ est atteinte par un robot. Seul un robot arrivant par l'entrée v_i peut accéder à la sortie v'_i.

Démonstration. Si $\forall v_j \in N(v_i)$, l'entrée $e_{v_i,\neg v_j}$ est atteinte par un robot, alors r_i peut s'appuyer sur eux les uns après les autres pour accéder à la sortie v'_i. Considérons qu'aucun robot n'atteigne l'entrée $e_{v_i,\neg v_j}$, alors r_i sera bloqué dans les cases marquées par $\times$ dans l'étage correspondant. Dans ce cas, r_i ne sera pas en capacité d'atteindre la sortie v'_i. ■

Tout comme dans la preuve précédente, un robot qui accède à une entrée $e_{v_i,\neg v_j}$ du gadget v_i-SOMMET peut aider le robot venant de l'entrée v_i avant de ressortir du gadget par l'entrée d'où il provient. Un robot qui rentre dans un gadget v_i-SOMMET ne peut pas sortir par une autre entrée que celle d'où il arrive (s'il essaye, il sera bloqué dans les cases marquées par $\times$).

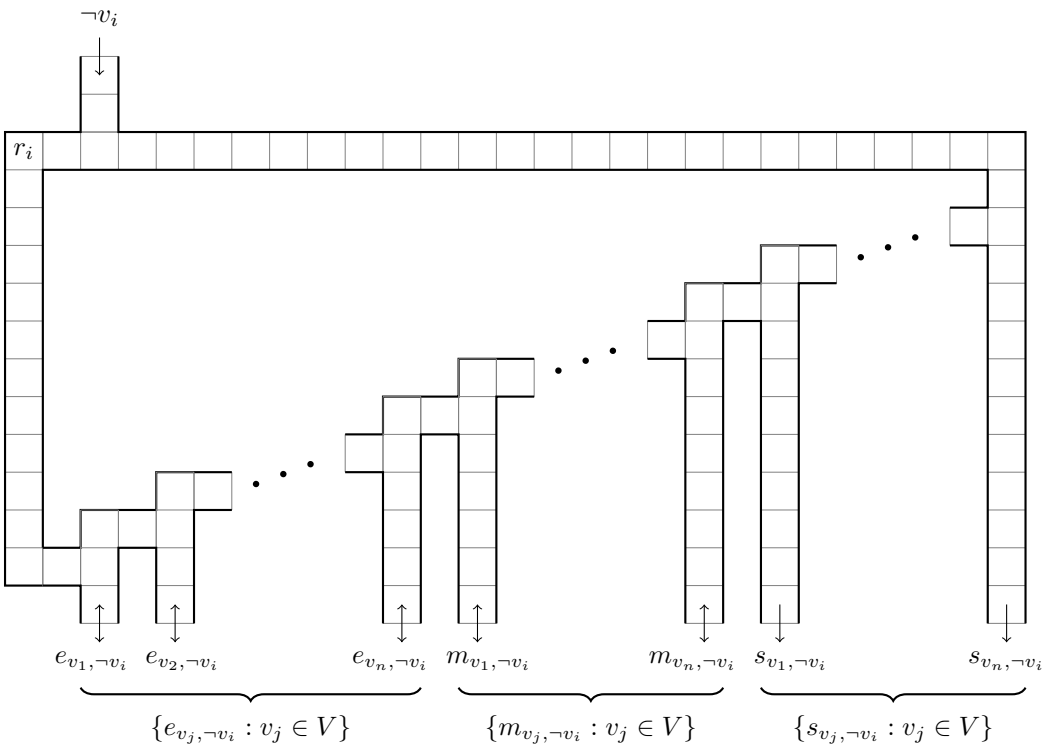


FIGURE 5.13 – *Gadget* $\neg v_i$-ROUTEUR. *Un gadget* $\neg v_i$-ROUTEUR *possède* $3|V|$ *sorties, qu'il peut également utiliser en entrée. Un robot entrant dans un* $\neg v_i$-ROUTEUR *peut toujours atteindre la sortie qu'il souhaite, peu importe d'où il vient. Toutefois il ne peut pas accéder à* $\neg v_i$ *(il s'agit d'une entrée). Le robot* r_i *a pour position de départ la case marquée* r_i *si et seulement si le sommet correspondant* v_i *n'appartient pas à l'ensemble indépendant de départ* A *(si et seulement si* $v_i \notin A$*).*

Le gadget v_i-DÉPLACEMENT (voir figure 5.16) est un gadget utilisé pour vérifier l'ensemble des positions des robots avant que l'un d'entre eux ne commence un déplacement (i.e. un passage d'un ensemble indépendant à un nouvel ensemble). Ce gadget, proche du gadget v_i-SOMMET, propose la caractéristique suivante :

Propriété 7 : [Propriété de v_i-DÉPLACEMENT] Dans un gadget v_i-DÉPLACEMENT, seul un robot venant de l'entrée M_{v_i} peut accéder à la sortie d_{v_i} et seulement si $\forall v_j \in V$ tel que $v_i \neq v_j$ un robot accède au gadget soit par l'entrée m_{v_i,v_j} soit par l'entrée $m_{v_i,\neg v_j}$.

Autrement dit, à chaque « étage » du gadget, le robot venant de l'entrée M_{v_i} doit être aidé par un autre robot venant d'une entrée à sa droite ou à sa gauche pour atteindre l'étage suivant. Dans la construction que nous proposons, $\forall v_j \in V$, le même robot est utilisé pour accéder aux entrées m_{v_i,v_j} et $m_{v_i,\neg v_j}$ (en fonction de si le robot représentant le sommet v_j appartient à l'ensemble indépendant ou non). Il est donc impossible que deux robots accèdent aux entrées m_{v_i,v_j} et $m_{v_i,\neg v_j}$ en même temps, ils ne peuvent pas s'aider pour changer d'étage et donc seul le robot venant de l'entrée M_{v_i} peut accéder à la sortie d_{v_i}.

Nous appelons le gadget suivant $(v_i, \neg v_j)$-SLIDING (voir figure 5.17). Ce gadget est utilisé pour simuler le passage d'un ensemble indépendant s_l à l'ensemble suivant s_{l+1} avec $s_{l+1} = \{v_j\} \cup s_l \backslash \{v_i\}$. Autrement dit, s_{l+1} correspond à s_l duquel on retire v_i et on ajoute v_j.

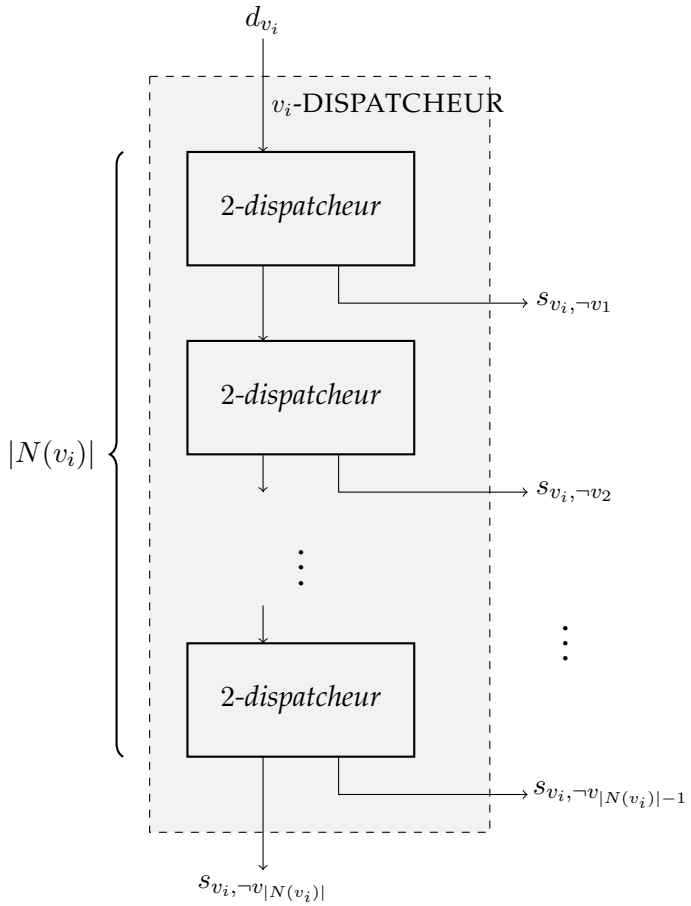


FIGURE 5.14 – *Gadget* v_i-DISPATCHEUR. *Construction d'un gadget* v_i-DISPATCHEUR *(correspondant à un* $|N(v_i)|$*-dispatcher) à partir de* $|N(v_i)| - 1$ *gadgets 2-dispatcheur. Pour chaque* $v_j \in N(v_i)$*, il existe une sortie* $s_{v_i,\neg v_j}$*.*

Propriété 8 : [Propriété de $(v_i, \neg v_j)$-SLIDING] Dans un gadget $(v_i, \neg v_j)$-SLIDING (voir figure 5.17), considérant deux robots r_i et r_j arrivant respectivement par les entrées v_i et $\neg v_j$, le robot r_i peut seulement accéder à la sortie $\neg v_i$ et le robot r_j peut seulement accéder à la sortie v_j.

Si un robot est seul dans le gadget, il est bloqué par les cases marquées $\times$. Considérant deux robots r_i et r_j arrivant respectivement par les entrées v_i et $\neg v_j$, ils peuvent collaborer afin d'accéder tous les deux à une sortie. Les robots doivent réaliser les mouvements suivants (autrement ils seront bloqués par les cases marquées $\times$) : r_i : $\rightarrow, \downarrow, \leftarrow$; r_j : $\rightarrow, \downarrow, \leftarrow, \downarrow, \rightarrow, \uparrow$; r_i : $\rightarrow, \downarrow$; r_j : $\downarrow, \rightarrow$. Avec ces mouvements, r_i accède à la sortie $\neg v_i$ et r_j accède à la sortie v_j, ils ne peuvent pas échanger leurs sorties.

Le dernier gadget utilisé est le gadget CIBLE (figure 5.18) qui contient l'unique case d'arrivée. À noter que la case d'arrivée n'est accessible que si k robots accèdent au gadget CIBLE. Un robot qui accède au gadget CIBLE ne peut plus en sortir.

Avant de présenter la construction, nous introduisons l'idée principale du fonctionnement de l'instance que nous créons.

Remarque 1. Dans la construction suivante, à chaque sommet $v_i \in V$ correspond un ro-

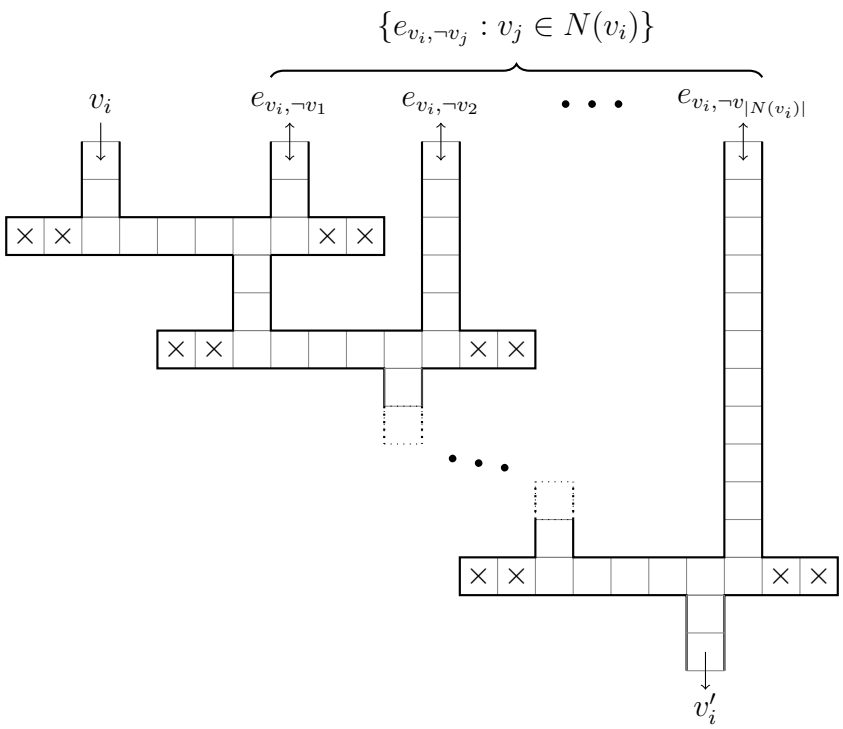


FIGURE 5.15 – *Gadget v_i-*SOMMET*. Considérant un sommet $v_i \in V$, le gadget v_i-*SOMMET *correspondant est constitué de $|N(v_i)| + 1$ entrées. Seul le robot venant de l'entrée v_i peut atteindre la sortie v'_i. Pour l'atteindre, il a besoin de l'aide successive de robots venant des entrées de type $e_{v_i,\neg v_j}$ ($\forall v_j \in N(v_i)$), autrement le robot venant de l'entrée v_i ne sera pas en mesure d'accéder à la sortie v'_i et sera bloqué dans les cases marquées par $\times$.*

bot r_i. À chaque étape, pour chaque robot r_i, si le sommet correspondant v_i appartient à l'ensemble indépendant actuel alors r_i est dans le gadget v_i-ROUTEUR, autrement il est dans le gadget $\neg v_i$-ROUTEUR.Soit s_l et s_{l+1} deux ensembles indépendants, de taille $|s_l| = |s_{l+1}| = k$ et $s_{l+1} = \{v_j\} \cup s_l \backslash \{v_i\}$. Les robots r_i et r_j se situent respectivement dans les gadgets v_i-ROUTEUR et $\neg v_j$-ROUTEUR. Pour passer de s_l à s_{l+1}, le robot r_i accède au gadget v_i-DÉPLACEMENT afin de vérifier que les autres robots ne soient pas déjà en train de modifier s_l (afin d'empêcher l'instance de déplacer deux sommets en même temps). Ainsi, r_i accède au gadget v_i-DISPATCHEUR afin de se diriger vers le gadget $(v_i, \neg v_j)$-SLIDING, r_j sort du gadget $\neg v_j$-ROUTEUR et le rejoint. Ensuite, r_i accède au gadget $\neg v_i$-ROUTEUR et r_j au gadget v_j-SOMMET afin de vérifier que l'ensemble des voisins de v_j ($N(v_j)$) ne sont pas dans le nouvel ensemble ainsi créé (i.e. que l'ensemble obtenu est un ensemble indépendant). Si le nouvel ensemble est un ensemble indépendant, v_j accède au gadget v_j-ROUTEUR, autrement il reste bloqué dans le gadget v_j-SOMMET. À chaque étape, nous avons un ensemble indépendant de taille k et lorsque tous les robots représentant les sommets de B sont dans leur gadget v_i-ROUTEUR respectif, ils peuvent rejoindre le gadget CIBLE, accéder à la case d'arrivée et donc à une configuration gagnante.

Construction 2 : Soit I une instance de TOKEN SLIDING avec un graphe $G = (V, E)$ avec $|V| = n$ et deux ensembles indépendants A et B de k sommets. L'instance correspondante I' du problème d'ACCESSIBILITÉ GÉNÉRALE est construite à partir

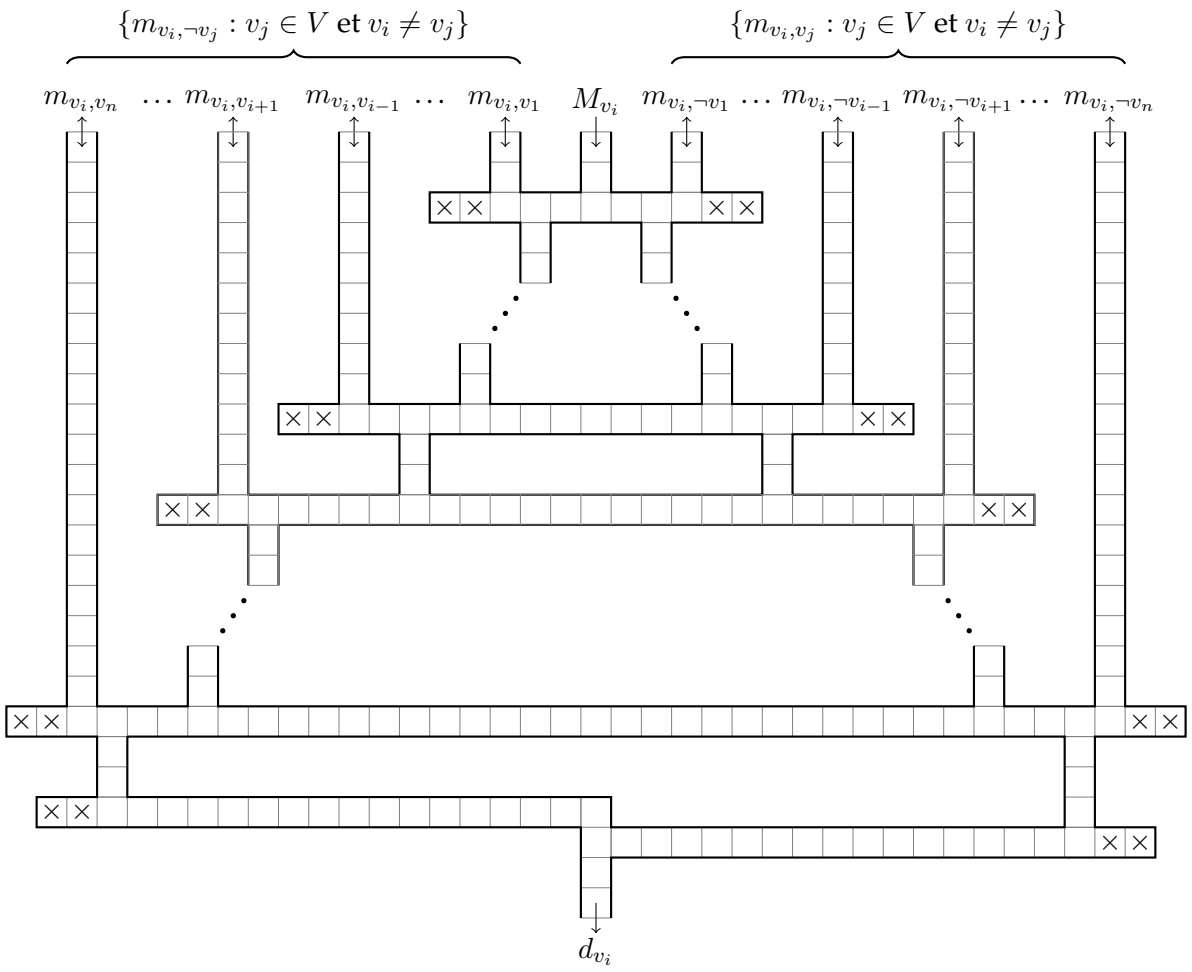


FIGURE 5.16 – *Gadget v_i-*DÉPLACEMENT*. Un robot r_i arrivant par l'entrée M_{v_i} peut accéder à la sortie d_{v_i} si et seulement si, pour chaque étage, un autre robot l'aide en accédant à l'entrée m_{v_i,v_j} ou $m_{v_i,\neg v_j}$. Autrement, r_i sera bloqué dans le gadget par les cases marquées $\times$. Dans notre construction, il est impossible que deux robots accèdent aux entrées m_{v_i,v_j} et $m_{v_i,\neg v_j}$ en même temps.*

des gadgets suivants :

- $\forall v_i \in V$ nous considérons un robot $r_i \in R$.
- Les n robots $r_i \in R$ sont de la même couleur.
- $\forall r_i \in R$ (et donc $\forall v_i \in V$) nous construisons un gadget v_i-ROUTEUR, un gadget $\neg v_i$-ROUTEUR, un gadget v_i-DÉPLACEMENT, un gadget v_i-SOMMET avec $|N(v_i)|$ entrés de type $e_{v_i,\neg v_j}$ et un gadget v_i-DISPATCHEUR de $|N(v_i)|$ sorties (soit un dispatcheur de $|N(v_i)|$ sorties).
- $\forall (v_i, v_j) \in E$ nous construisons un gadget $(v_i, \neg v_j)$-SLIDING et un gadget $(v_j, \neg v_i)$-SLIDING.
- $\forall r_i \in R$, si $v_i \in A$, alors r_i commence dans le gadget v_i-ROUTEUR (voir case marquée r_i dans la figure 5.12) sinon il commence dans le gadget $\neg v_i$-ROUTEUR (voir case marquée r_i dans la figure 5.13).

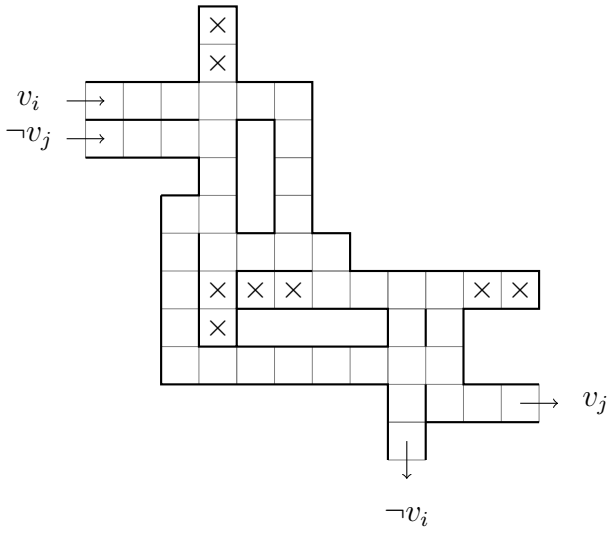


FIGURE 5.17 – *Gadget $(v_i, \neg v_j)$-SLIDING. Considérant deux robots r_i et r_j arrivant respectivement des entrées v_i et $\neg v_j$, en accord avec la propriété 8, r_i peut seulement accéder à la sortie $\neg v_i$ et r_j seulement à la sortie v_j.*

– Nous construisons un gadget CIBLE avec $k = |A| = |B|$ qui contient l'unique case d'arrivée t. Donc l'ensemble des cases d'arrivée de I' correspond à $T = \{t\}$.

Les gadgets, $\forall v_i \in V$, sont connectés comme suit :

– **Connexions des sorties du gadget $\neg v_i$-ROUTEUR :** pour tout $v_j \in V$ et $v_j \neq v_i$, si $(v_j, v_i) \in E$, la sortie $e_{v_j,\neg v_i}$ (resp. $s_{v_j,\neg v_i}$) est connecté à l'entrée $e_{v_j,\neg v_i}$ (respectivement $\neg v_i$) du gadget v_j-ROUTEUR (resp. $(v_j, \neg v_i)$-SLIDING). Si $(v_j, v_i) \notin E$ alors les sorties $e_{v_j,\neg v_i}$ et $s_{v_j,\neg v_i}$ sont fermées par un mur. La sortie $m_{v_j,\neg v_i}$ est connectée à l'entrée $m_{v_j,\neg v_i}$ du gadget v_j-DÉPLACEMENT.

– **Connexions des sorties du gadget v_i-SOMMET :** la sortie v'_i est connectée à l'entrée v'_i du gadget v_i-ROUTEUR.

– **Connexions des sorties du gadget v_i-ROUTEUR :** la sortie M_{v_i} est connectée à l'entrée M_{v_i} du gadget v_i-DÉPLACEMENT. Pour tout $v_j \in V$ et $v_j \neq v_i$, la sortie m_{v_j,v_i} est connectée à l'entrée m_{v_j,v_i} du gadget v_j-DÉPLACEMENT. Si $v_i \in B$ alors la sortie T_{v_i} est connectée à l'entrée T_{v_i} du gadget CIBLE, sinon, la sortie est fermée par un mur.

– **Connexions des sorties du gadget v_i-DÉPLACEMENT :** la sortie d_{v_i} est connectée à l'entrée d_{v_i} du gadget v_i-DISPATCHEUR.

– **Connexions des sorties du gadget v_i-DISPATCHEUR :** pour tout $v_j \in N(v_i)$, la sortie $s_{v_i,\neg v_j}$ est connectée à l'entrée v_i du gadget $(v_i, \neg v_j)$-SLIDING.

– **Connexions des sorties du gadget $(v_i, \neg v_j)$-SLIDING :** la sortie $\neg v_i$ (resp. v_j) est connectée à l'entrée $\neg v_i$ (resp. v_j) du gadget $\neg v_i$-ROUTEUR (resp. v_j-ROUTEUR).

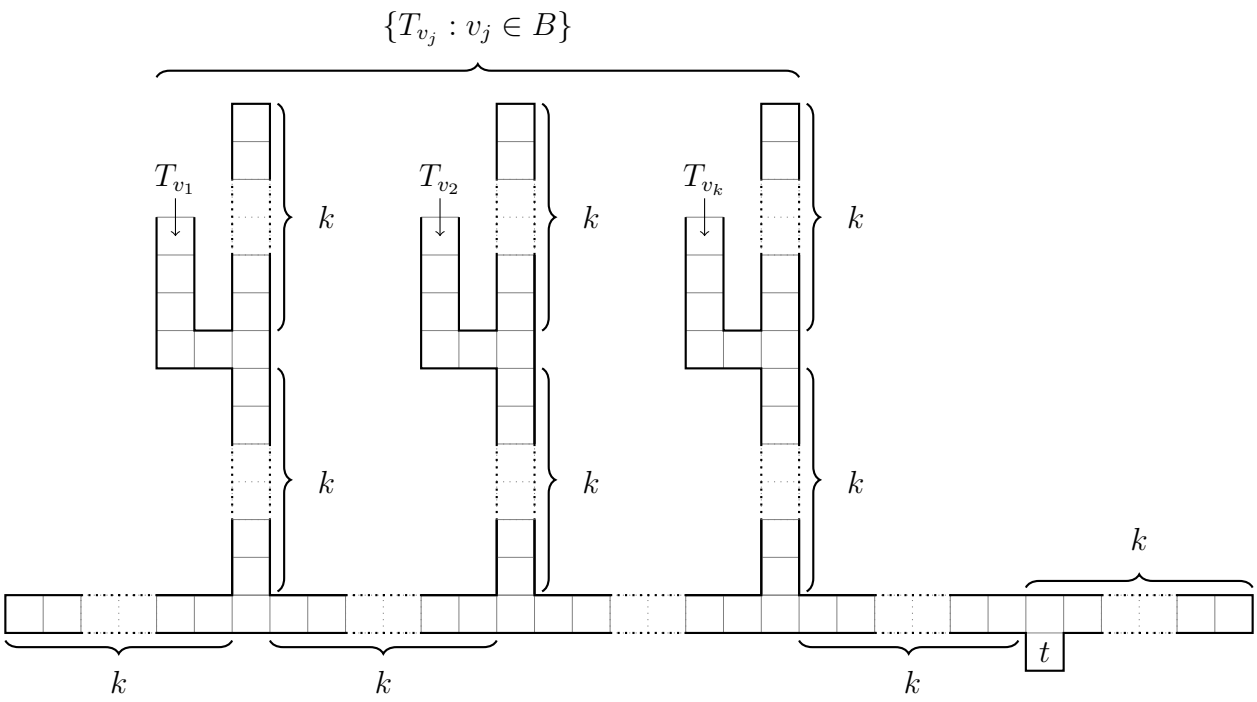


FIGURE 5.18 – *Gadget* CIBLE. *Le gadget* CIBLE *contient l'unique case d'arrivée t qui est accessible seulement si k robots accèdent au gadget.*

La figure 5.19 présente un exemple de la construction 2. Dans le but d'en faciliter sa compréhension, nous pouvons définir chaque niveau de l'instance I' que présente la figure 5.19 comme suit :

- (a): si un robot est dans un gadget $\neg v_i$-ROUTEUR alors v_i n'appartient pas à l'ensemble indépendant courant (i.e. les gadgets de (a) représentent l'ensemble des sommets qui n'appartiennent pas à l'ensemble indépendant courant).

- (b): permet de tester si l'opération de sliding réalisée est légale (i.e. si le nouvel ensemble obtenu est bien un ensemble indépendant).

- (c): contrairement au niveau (a), le niveau (c) représente les sommets de l'ensemble indépendant actuel.

- (d): vérifie qu'un robot ne puisse pas modifier l'ensemble indépendant actuel avant que l'opération de sliding précédent ne soit finie. Si un robot traverse ce niveau pour accéder au niveau (e) alors aucun autre robot ne peut le traverser avant que l'opération de sliding précédente ne soit terminée et validée par le niveau (b).

- (e): en traversant le niveau (d), le robot r_i choisit le sommet qui rentre dans le nouvel ensemble. Par exemple, pour réaliser l'opération de sliding qui retire le sommet v_i de l'ensemble courant et ajoute v_j, le robot r_i doit traverser le gadget v_i-DISPATCHEUR et choisir la sortie connectée au gadget $(v_i, \neg v_j)$-SLIDING.

- (f): est utilisé afin de simuler une opération de sliding. En traversant ce niveau, le robot r_i accède au gadget $\neg v_i$-ROUTEUR de l'étage (a) (i.e. v_i n'appartient plus à l'ensemble courant) et r_j accède au niveau (b) afin de vérifier si le nouvel ensemble est

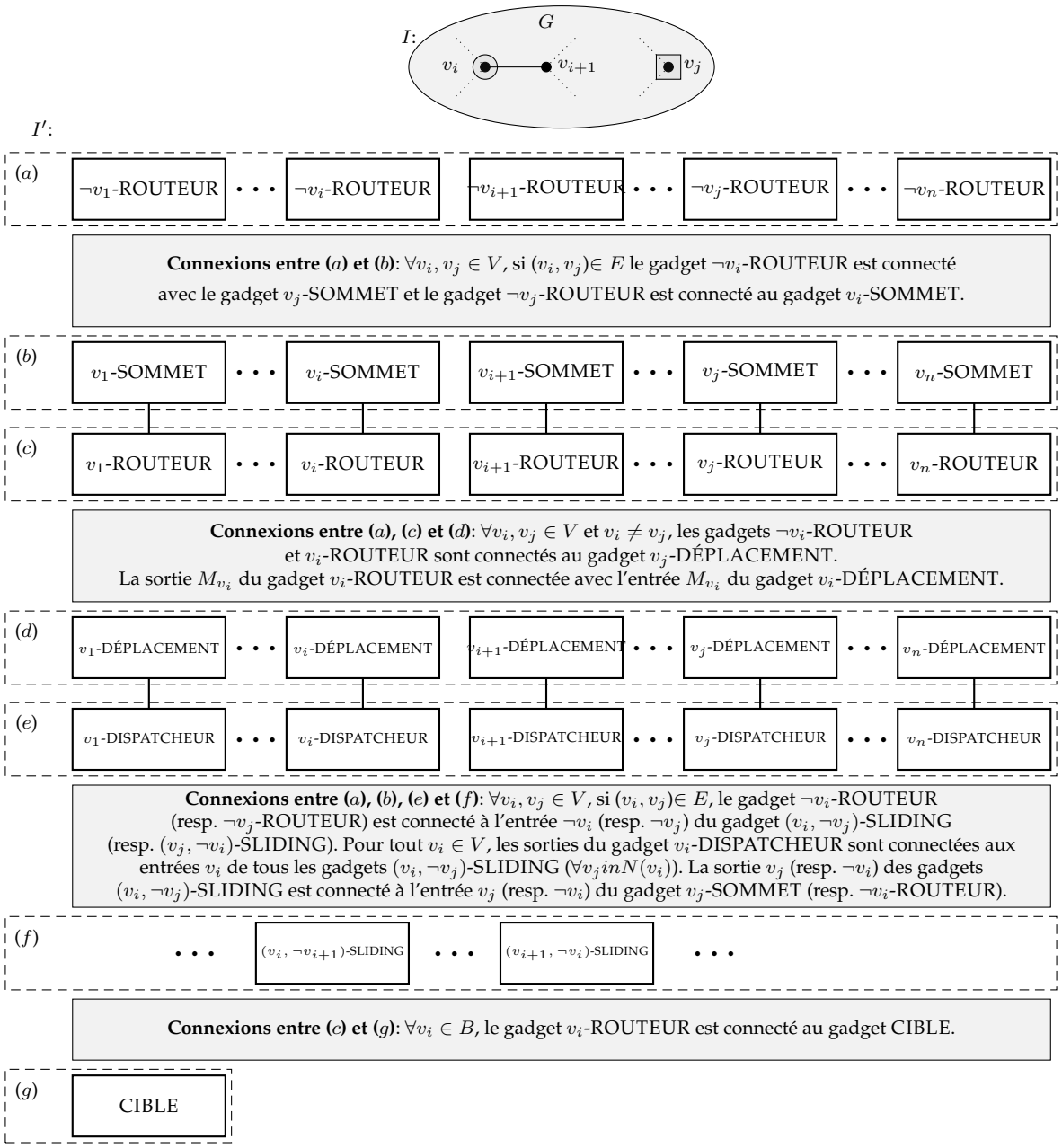


FIGURE 5.19 – *Exemple d'instance I' du problème d'*ACCESSIBILITÉ GÉNÉRALE *obtenue par la construction 2 depuis une instance I de* TOKEN SLIDING*. Dans l'instance I, les sommets marqués par un cercle sont les sommets dans l'ensemble indépendant courant (initialement, les sommets de A). Les sommets marqués par des carrés sont les sommets de l'ensemble indépendant objectif, B.*

un ensemble indépendant et s'il l'est, r_j accède au gadget v_j-ROUTEUR de l'étage (c) (i.e. v_j est ajouté au nouvel ensemble indépendant).

Afin de simplifier la preuve montrant que la construction 2 est une réduction de polynomiale de TOKEN SLIDING vers le problème d'ACCESSIBILITÉ GÉNÉRALE, nous commençons par quelques remarques et caractéristiques que possèdent l'instance I' du pro-

blème d'ACCESSIBILITÉ GÉNÉRALE obtenue à partir d'une instance I de TOKEN SLIDING.

Remarque 2. Dans une instance obtenue par construction 2, pour quitter un gadget v_i-ROUTEUR et accéder au gadget $\neg v_i$-ROUTEUR, le robot r_i doit commencer par prendre la sortie M_{v_i} et accéder au gadget v_i-DÉPLACEMENT. D'après la propriété 7, si r_i reçoit l'aide des autres robots, il peut traverser ce gadget et accéder à la sortie d_{v_i} puis au gadget v_i-DISPATCHEUR. Ensuite, r_i accède à une sortie $s_{v_i,\neg v_j}$ et rentre dans le gadget $(v_i, \neg v_j)$-SLIDING correspondant par l'entrée v_i. En accord avec la propriété 8 et avec l'aide d'un autre robot r_j arrivant par l'entrée $\neg v_j$, r_i peut atteindre la sortie $\neg v_i$ et accéder au gadget $\neg v_i$-ROUTEUR.

Remarque 3. Dans une instance obtenue par construction 2, pour quitter un gadget $\neg v_j$-ROUTEUR et accéder au gadget v_j-ROUTEUR, le robot r_j doit commencer par prendre la sortie $s_{v_i,\neg v_j}$ et accéder au gadget $(v_i, \neg v_j)$-SLIDING par l'entrée $\neg v_j$. D'après la propriété 8, un robot r_i doit le rejoindre dans ce gadget par l'entrée v_i afin de l'aider à atteindre la sortie v_j et le gadget v_j-SOMMET. En accord avec la propriété 6, r_j peut traverser le gadget v_j-SOMMET et atteindre le gadget $\neg v_j$-ROUTEUR si et seulement si pour chaque $v_k \in N(v_j)$, le robot correspondant r_k peut aider r_j (i.e. si et seulement si $\forall v_k \in N(v_i)$, le robot r_k est situé dans le gadget $\neg v_k$-ROUTEUR).

Remarque 4. Conformément à la remarque 2, lorsqu'un robot r_i passe du gadget v_i-ROUTEUR au gadget $\neg v_i$-ROUTEUR, un autre robot r_j (tel que $v_j \in N(v_i)$) réalise l'inverse (il passe du gadget $\neg v_j$-ROUTEUR au gadget v_j-ROUTEUR). De plus, d'après la remarque 3, lorsqu'un robot r_i passe du gadget $\neg v_i$-ROUTEUR au gadget v_i-ROUTEUR, un robot r_j (tel que $v_j \in N(v_i)$) réalise également l'inverse (il passe du gadget v_j-ROUTEUR au gadget $\neg v_j$-ROUTEUR). De la remarque 2 et la remarque 3 on peut en conclure qu'un robot r_i peut passer du gadget v_i-ROUTEUR au gadget $\neg v_i$-ROUTEUR si et seulement si un robot r_j (avec $v_j \in N(v_i)$) passe du gadget $\neg v_j$-ROUTEUR au gadget v_j-ROUTEUR.

Remarque 5. Dans toute instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE obtenue par construction 2, $\forall v_i \in V$, seul le robot correspondant r_i peut accéder aux gadgets v_i-ROUTEUR et $\neg v_i$-ROUTEUR. En effet, considérant un robot r_i dans le gadget v_i-ROUTEUR, si r_i essaie d'accéder à un gadget $\neg v_j$-ROUTEUR (avec $v_j \neq v_i$), il doit accéder à la sortie $\neg v_j$ d'un gadget $(v_j, \neg v_k)$-SLIDING. Or, les propriétés 6, 7 et 8 impliquent que les seuls gadgets accessibles à r_i sont les gadgets décrits dans la remarque 2 et le gadget CIBLE (seulement si $v_i \in B$). Par les mêmes arguments, un robot r_i ne peut pas passer du gadget $\neg v_i$-ROUTEUR au gadget v_j-ROUTEUR (avec $v_i \neq v_j$).

Remarque 6. Un robot r_i arrivant dans le gadget v_i-DÉPLACEMENT par l'entrée M_{v_i} peut accéder à la sortie d_{v_i} si et seulement si $\forall v_j \in V \backslash \{v_i\}$, le robot r_j a accès soit au gadget v_j-ROUTEUR soit au gadget $\neg v_j$-ROUTEUR. En effet, d'après la propriété 7, la sortie d_{v_i} peut être atteinte par r_i si et seulement si un autre robot l'aide à chaque étage. De plus, par la construction 2, chaque entrée m_{v_i,v_j} est connectée à une sortie du gadget v_j-ROUTEUR et l'entrée opposée $m_{v_i,\neg v_j}$ à une sortie du gadget $\neg v_j$-ROUTEUR. Donc,

pour que r_i accède à la sortie d_{v_i}, il est nécessaire que $\forall v_j \in V \backslash \{v_i\}$, le robot r_j puisse accéder soit au gadget v_j-ROUTEUR soit au gadget $\neg v_j$-ROUTEUR.

Remarque 7. De la remarque 6, on peut ajouter que dans une instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE obtenue par construction 2, quand un robot r_i quitte le gadget v_i-ROUTEUR pour traverser le gadget v_i-DÉPLACEMENT, aucun autre robot r_k ne peut franchir le gadget v_k-DÉPLACEMENT tant que r_i n'a pas accédé au gadget $\neg v_i$-ROUTEUR et tant que r_j n'a pas quitté le gadget $\neg v_j$-ROUTEUR pour accéder au gadget v_j-ROUTEUR (conformément à la remarque 4). Pour s'en convaincre, prenons le cas où deux robots r_i et r_j tentent de franchir respectivement les gadgets v_i-DÉPLACEMENT et v_j-DÉPLACEMENT. D'après la propriété 7 et la remarque 6 r_i (resp. r_j) ne peut pas accéder à la sortie d_{v_i} (resp. d_{v_j}) car il a besoin de l'aide de r_j (resp. r_i). De plus, si r_j aide r_i à accéder à la sortie d_{v_i} du gadget v_i-DÉPLACEMENT avant d'essayer de franchir le gadget v_j-DÉPLACEMENT, il sera bloqué dans le gadget tant que r_i n'est pas arrivé au gadget $\neg v_i$-ROUTEUR.

Théorème 6 : Le problème d'ACCESSIBILITÉ GÉNÉRALE est **PSPACE**-complet.

Démonstration. L'argument pour dire que le problème appartient à **PSPACE** est proche de celui donné par Hüffner [37] pour le problème ATOMIX. Étant donné une instance du problème d'ACCESSIBILITÉ GÉNÉRALE de Ricochet Robots avec un plateau de jeu P constitué de $|C|$ cases, une machine de Turing non déterministe peut la résoudre en appliquant un mouvement de manière répétée jusqu'à atteindre une configuration gagnante. Le nombre de configurations étant borné supérieurement par $\binom{|C|}{n}$, la machine peut déclarer qu'il n'y a pas de solution après avoir réalisé plus de mouvement sans trouver de configuration gagnante. De plus, puisque les instance du problème d'ACCESSIBILITÉ GÉNÉRALE sont encodablents en espace polynomial, le problème appartient à **NPSPACE** et donc, d'après le théorème de Savitch [67], le problème appartient à **PSPACE**.

Maintenant, nous devons prouver qu'il existe une réduction polynomiale de TOKEN SLIDING vers le problème d'ACCESSIBILITÉ GÉNÉRALE. Considérant une instance I de TOKEN SLIDING et l'instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE obtenue par la construction 2 à partir de l'instance I :

– Supposons qu'il existe une solution positive à l'instance I de TOKEN SLIDING, une solution positive pour l'instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE est construite comme suit :

  Soit S un certificat positif de l'instance I.Alors S est une séquence d'ensembles indépendants (numérotés de $s_1, \ldots, s_{|S|}$) tel que s_{l+1} est obtenu en retirant un sommet de s_l et en ajoutant un de ses voisins dans G et tel que $s_1 = A$ et $s_{|S|} = B$. Par construction, $\forall v_i \in V$, si $v_i \in A$, le robot correspondant r_i commence dans le gadget v_i-ROUTEUR, sinon il commence dans le gadget $\neg v_i$-ROUTEUR. La position de départ des robots correspond à l'ensemble indépendant A.

Considérons deux ensembles indépendants $s_l \in S$ et $s_{l+1} \in S$, tel que s_{l+1} est obtenu en retirant v_i de s_l et en y ajoutant v_j (avec $(v_i, v_j) \in E$). Puisque $v_i \in s_l$, le robot r_i est dans le gadget v_i-ROUTEUR. Afin d'atteindre la configuration correspondante à s_{l+1} le robot r_i (correspondant au sommet v_i) doit accéder au gadget $\neg v_i$-ROUTEUR et le robot r_j (correspondant au sommet v_j) doit accéder au gadget v_j-ROUTEUR. Le robot doit donc réaliser les mouvements décrits par la remarque 2 et r_j ceux décrits par la remarque 3. Puisque s_{l+1} est un ensemble indépendant, $\forall v_h \in N(v_j)$, le robot correspondant r_h peut accéder au gadget $\neg v_h$-ROUTEUR et aider r_j à atteindre la sortie v'_j du gadget v_j-SOMMET afin d'accéder au gadget v_j-ROUTEUR. On arrive ainsi à la configuration correspondante à s_l à la différence près que v_i est passé du gadget v_i-ROUTEUR au gadget $\neg v_i$-ROUTEUR et le robot r_j est passé du gadget $\neg v_j$-ROUTEUR au gadget v_j-ROUTEUR, soit la configuration correspondante à s_{l+1}.

En répétant cette opération, il est possible de passer de la configuration correspondante à $s_1 = A$ à celle correspondante à $s_{|S|} = B$. Ainsi, nous arrivons à une configuration telle que $\forall v_i \in B$, le robot r_i est dans le gadget v_i-ROUTEUR. Par construction 2 ces $k = |B|$ robots peuvent accéder au gadget CIBLE et l'un d'entre eux peut atteindre la case d'arrivée t, on a donc atteint une configuration gagnante. Donc, si I admet une configuration positive, alors I' admet également une configuration positive.

– Réciproquement, supposons qu'il existe une solution positive à l'instance I' du problème d'ACCESSIBILITÉ GÉNÉRALE, alors nous construisons une solution positive pour l'instance I de TOKEN SLIDING comme suit :

L'instance I' admet une solution positive, donc k robots peuvent accéder au gadget CIBLE (sinon aucun robot ne peut accéder à la case d'arrivée). Donc, d'après la remarque 5 et la remarque 7 $\exists C$ une séquence de configurations de I' tel que $\forall c_l \in C$, $\forall r_i \in R$, r_i est soit dans le gadget v_i-ROUTEUR soit dans le gadget $\neg v_i$-ROUTEUR. À noter que la configuration c_1 correspond à la configuration de départ ($\forall v_i \in V$, si $v_i \in A$, le robot correspondant r_i commence dans le gadget v_i-ROUTEUR, sinon il commence dans le gadget $\neg v_i$-ROUTEUR).

La configuration $c_{|c|}$ est la configuration finale tel que $\forall v_i \in V$, si $v_i \in B$, le robot correspondant r_i se situe dans le gadget v_i-ROUTEUR sinon il est dans le gadget $\neg v_i$-ROUTEUR. Conformément à la remarque 7 deux robots ne peuvent pas traverser un gadget de type v_i-DÉPLACEMENT en même temps. D'après la propriété 6 et la construction 2, pour accéder au gadget v_j-ROUTEUR, le robot r_j doit traverser le gadget v_j-SOMMET et donc $\forall v_h \in N(v_j)$, le robot r_h est dans le gadget $\neg v_h$-ROUTEUR. Soit S une séquence d'ensembles de sommets tel que $\forall c_l \in C$, on considère l'ensemble correspondant $s_l \in S$ tel que $\forall r_i \in R$, si r_i est dans le gadget v_i-ROUTEUR dans la configuration c_l, alors $v_i \in s_l$. Alors, d'après la propriété 6 et la construction 2, $\forall s_l \in S$, l'ensemble s_l est un ensemble indépendant.

Par construction, un robot r_i peut se déplacer d'un gadget v_i-ROUTEUR au gadget $\neg v_i$-ROUTEUR si un autre robot r_j se déplace du gadget $\neg v_j$-ROUTEUR au gadget v_j-ROUTEUR et seulement si le gadget $(v_i, \neg v_j)$-SLIDING existe (donc seulement si $(v_i, v_j) \in E$). Donc $\forall s_l \in S$, l'ensemble s_l correspond à un ensemble indépendant de sommets tels que s_{l+1} est obtenu en retirant un sommet v_i de s_l et en y ajoutant

un sommet $v_j \in N_(v_i)$ (i.e. en réalisant une opération de sliding). De plus, $s_1 = A$ et $s_{|S|} = B$. Donc la séquence d'ensembles indépendants S correspondant à C est un certificat positif de l'instance I de TOKEN SLIDING.

Puisque la construction 2 peut être réalisée en temps polynomial le problème d'ACCESSIBILITÉ GÉNÉRALE est **PSPACE**-difficile. Par les arguments précédents, le problème d'ACCESSIBILITÉ GÉNÉRALE est **PSPACE**-complet. ■

5.5 Décidabilité du problème d'ACCESSIBILITÉ GÉNÉRALE

Dans cette section, nous montrons que le problème d'ACCESSIBILITÉ GÉNÉRALE devient indécidable si l'on considère des instances avec des plateaux de taille infinie et une infinité de robots. Afin de la prouver, nous montrons l'existence d'une réduction polynomiale entre machine de Turing et le problème d'ACCESSIBILITÉ GÉNÉRALE. En d'autres termes, nous montrons que le jeu de Ricochet Robot, avec un plateau de taille infinie et un nombre de robots infini, peut simuler toutes machine de Turing. L'énonce du théorème que nous prouvons dans cette sous-section est le suivant :

Théorème 7 : Le problème d'ACCESSIBILITÉ GÉNÉRALE d'un jeu de Ricochet Robots avec un plateau de taille infinie et un nombre infini de robots est indécidable.

Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ une machine de Turing de m états ($Q = \{q_0, \ldots, q_{m-1}\}$) et n symboles ($\Gamma = \{Y_0, \ldots, Y_{n-1}\}$). La construction que nous proposons est divisée principalement en deux gadgets :

- Le gadget RUBAN utilisé pour encoder les symboles écrits sur le ruban de M.
- Le gadget CONTRÔLEUR utilisé pour encoder l'état courant de M et permet de simuler sa fonction de transition d'états.

La sous-section suivante décrit comment les gadgets RUBAN et CONTRÔLEUR communiquent pour simuler les opérations de lecture et d'écriture. Ensuite, la sous-section 5.5.3 montre comment le gadget CONTRÔLEUR simule la fonction de transition suivant l'état actuel et le symbole lu par la tête de lecture. Enfin, la sous-section 5.5.4 montre que le problème d'ACCESSIBILITÉ GÉNÉRALE est Turing-complet.

5.5.1 Système de communication

Nous définissons un canal de communication comme un couloir infini (i.e. une ligne d'une case d'épaisseur, entourée par des murs) auquel on branche des connexions nécessitant l'aide d'un robot pour les atteindre (voir table 5.1). Le système de communication

entre le gadget RUBAN et le gadget CONTRÔLEUR est assuré par un gadget appelé gadget COMMUNICATION, composé de canaux de communication. La construction 3 décrit la construction du gadget COMMUNICATION.

Construction 3 : [Gadget COMMUNICATION] Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ une machine de Turing de m états ($Q = \{q_0, \ldots, q_{m-1}\}$) et n symboles ($\Gamma = \{Y_0, \ldots, Y_{n-1}\}$), le gadget COMMUNICATION est construit comme suit :

- Créer trois canaux de communication C_h, C'_L et C'_R.
- Pour chaque $Y_i \in \Gamma$ créer deux canaux de communication C_{Y_i} et C'_{Y_i}.

À noter que les canaux de communication ne peuvent être utilisés que dans un seul sens. La figure 5.20 représente le système de communication décrit dans la construction 3. Les canaux de communication C_{Y_i} ($\forall Y_i \in \Gamma$) sont utilisés par le gadget RUBAN pour transmettre le symbole de la case courante c_i au gadget CONTRÔLEUR. Les canaux de communication C'_{Y_i} ($\forall Y_i \in \Gamma$) sont utilisés par le CONTRÔLEUR pour transmettre au gadget RUBAN la nouvelle valeur à écrire dans la case c_i. Plus précisément, le gadget RUBAN (resp. CONTRÔLEUR) transmet un robot via le canal C_{Y_i} (resp. C'_{Y_i}) si la valeur courante (resp. la nouvelle valeur) de la case courante correspond au symbole Y_i. Le gadget CONTRÔLEUR indique au gadget RUBAN que la tête de lecture doit bouger vers la gauche (resp. la droite) en envoyant un robot via le canal C'_L (resp. C'_R). Le canal de communication C_h est utilisé par le gadget RUBAN afin de déclarer la fin de l'opération de lecture au gadget CONTRÔLEUR.

5.5.2 Ruban et tête de lecture/écriture

Dans cette section, nous présentons le gadget RUBAN qui est utilisé pour simuler à la fois le ruban et la tête de lecture/écriture de la machine de Turing.

Le gadget RUBAN est composé de plusieurs gadgets CELLULE dont la construction est définie ci-dessous.

Construction 4 : [Gadget RUBAN] Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ une machine de Turing de m états ($Q = \{q_0, \ldots, q_{m-1}\}$) et n symboles ($\Gamma = \{Y_0, \ldots, Y_{n-1}\}$). Étant donné un gadget COMMUNICATION obtenu par la construction 3, le gadget RUBAN (figure 5.21) est construit de la façon suivante.
Soit c_i la $i^{\text{ème}}$ cellule du ruban de M. Pour chaque cellule c_i, un gadget CELLULE $Cell_{c_i}$ (voir figure 5.21) est obtenu par la construction suivante :

- construire un gadget n-routeur (appelé R_{c_i}),
- construire un gadget $2(n+1)$-routeur (appelé R'_{c_i}),
- construire deux gadgets 2-synchronieur (appelés $Left_{c_i}$ et $Right_{c_i}$), et

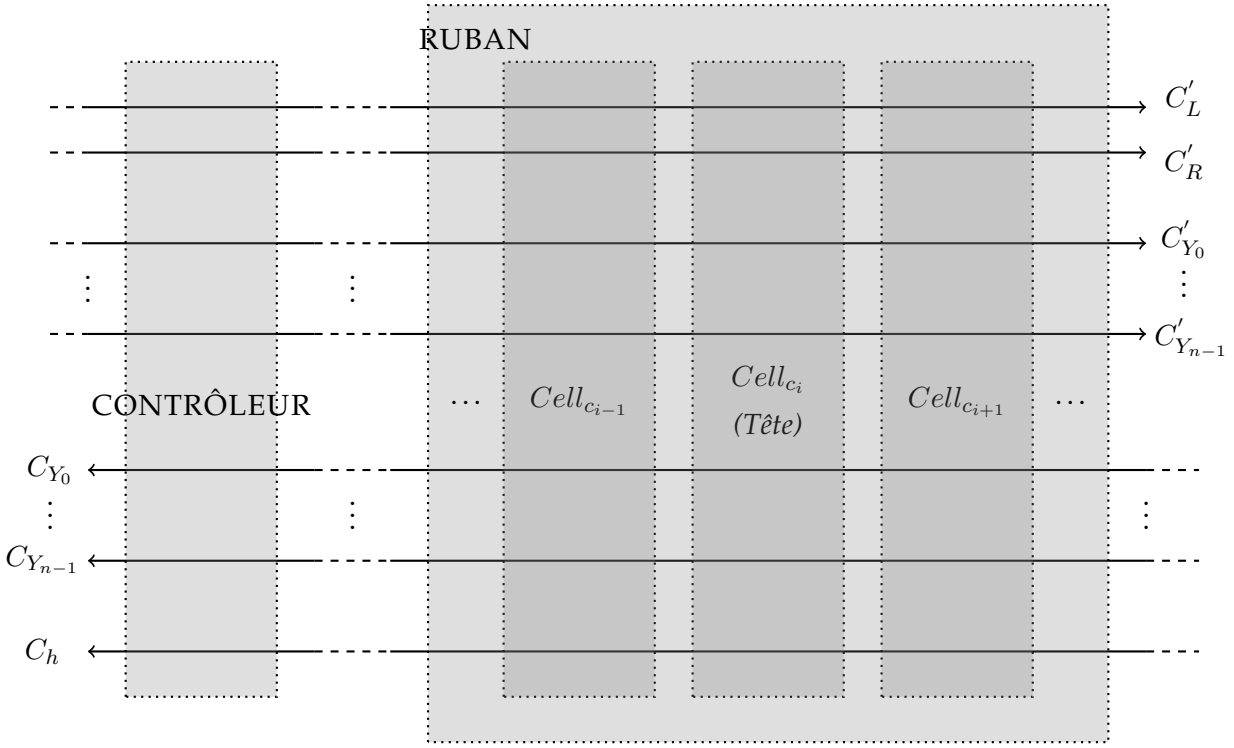


FIGURE 5.20 – *Gadget* COMMUNICATION. *Les flèches correspondent au sens dans lequel la communication s'opèrent, les valeurs correspondent au message envoyé. Par exemple, le canal de communication C_{Y_0} est utilisé pour envoyer le message « Y_0 » depuis le gadget* RUBAN *vers le gadget* CONTRÔLEUR. *À noter qu'un canal de communication ne peut être utilisé que dans un seul sens.*

– pour chaque symbole $Y_j \in \Gamma$, construire deux 2-synchronieur gadgets (appelés $Y_j^{c_i}$ et $Write(Y_j^{c_i})$).

Les éléments d'un gadget CELLULE $Cell_{c_i}$ sont connectés entre eux comme suit (voir figure 5.21) :

– le gadget R'_{c_i} intersecte le canal C'_L vers le gadget $Left_{c_i}$,

– le gadget R'_{c_i} intersecte C'_R vers le gadget $Right_{c_i}$,

– pour chaque $Y_j \in \Gamma$, une sortie du gadget $Write(Y_j^{c_i})$ et une sortie du gadget R_{c_i} sont connectées aux entrées du gadget $Y_j^{c_i}$. Une sortie de $Y_j^{c_i}$ est connectée au canal de communication C_{Y_j} et l'autre sortie au canal C_h, et

– pour chaque $Y_j \in \Gamma$, le gadget R'_{c_i} intersecte le canal C'_{Y_j} vers le gadget $Write(Y_j^{c_i})$, une sortie de R'_{c_i} est connectée à une entrée de $Write(Y_j^{c_i})$ et la dernière sortie disponible du gadget $Write(Y_j^{c_i})$ est connectée en même temps avec une entrée du gadget $Left_{c_i}$ et une entrée de $Right_{c_i}$.

Pour finir, pour chaque gadget $Cell_{c_i}$, le gadget est connecté aux gadgets $Cell_{c_{i-1}}$ et $Cell_{c_{i+1}}$ de la façon suivante :

– une des deux sorties du gadget $Left_{c_i}$ est connectée à l'entrée du gadget $R_{c_{i-1}}$, l'autre à l'entrée du gadget $R'_{c_{i-1}}$, et

– une des deux sorties du gadget $Right_{c_i}$ est connectée à l'entrée du gadget $R_{c_{i+1}}$, l'autre à l'entrée du gadget $R'_{c_{i+1}}$.

Dans une machine de Turing, la tête doit être en mesure d'accomplir les trois actions suivantes (à condition que la fonction de transition d'états le permette) :

– lire le symbole de la case courante,

– écrire un symbole dans la case courante,

– se déplacer sur le ruban vers la gauche ou vers la droite.

La figure 5.21 présente un gadget CELLULE $Cell_{c_i}$ (les couleurs représentent différents scénarios en fonction des actions que la tête de lecture doit réaliser, ces scénarios sont décrits ci-après). Dans chaque gadget CELLULE $Cell_{c_i}$, un robot r_{c_i} est utilisé pour encoder le symbole écrit dans la cellule correspondante c_i de M. Le robot r_{c_i} se trouve dans le gadget 2-synchronieur $Y_j^{c_i}$ si et seulement si la $i^{\text{ème}}$ cellule du ruban de M contient le symbole Y_j. Si le robot r_{c_i} est dans le gadget Y_j^i alors on dit que $Cell_i$ contient Y_j. Deux robots r_{h_1} et r_{h_2} sont utilisés afin de simuler la tête de M. Le robot r_{h_2} est dans le gadget $Cell_{c_i}$ si et seulement si la tête est sur la $i^{\text{ème}}$ cellule du ruban. Dans ce cas, le gadget $Cell_{c_i}$ est considéré comme le gadget CELLULE courant.

Lemme 2 : Soit un gadget CELLULE $Cell_{c_i}$ obtenu par construction 4.

1. Si les robots r_{c_i} et r_{h_1} sont dans les gadgets $Y_j^{c_i}$ et R_{c_i} respectivement, alors r_{c_i} et r_{h_1} ont accès aux canaux C_{Y_j} et C_h.
2. Si le robot r_{c_i} et r_{h_1} viennent du canal de communication des canaux de communication C'_{Y_j} et C'_L (resp. C'_R) et r_{h_2} se situe dans le gadget R'_{c_i}, alors r_{c_i} accède au gadget $Y_j^{c_i}$, r_{h_1} au gadget $R_{c_{i-1}}$ (resp. $R_{c_{i+1}}$) et r_{h_2} accède au gadget $R'_{c_{i-1}}$ (resp. $R'_{c_{i+1}}$).

Démonstration.

1. Si le robot r_{h_1} rentre dans un gadget $Y_k^{c_i}$ tel que $k \neq j$, alors, d'après la propriété des gadgets k-synchronieur (propriété 4), le robot r_{h_1} est bloqué dans le gadget $Y_k^{c_i}$ et r_{c_i} dans $Y_j^{c_i}$. Ainsi, supposons que r_{h_1} rentre dans le gadget $Y_j^{c_i}$. D'après la propriété 4, les robots r_{c_i} et r_{h_1} ont accès aux sorties du gadget $Y_j^{c_i}$, et ainsi, un des robots accède au canal de communication C_{Y_j} et l'autre au canal C_h.
2. D'après la propriété des gadgets k-routeur (propriété 3), r_{h_2} peut intercepter successivement les robots r_{c_i} et r_{h_1} pour leur permettre d'atteindre les gadgets $Write(Y_j^{c_i})$ and $Left_{c_i}$ (resp. $Right_{c_i}$) respectivement. Si r_{h_2} rentre dans un gadget $Write(Y_k^{c_i})$ tel que $k \neq j$, alors, d'après la propriété 4, $r_{c_i}, r_{h_1}, r_{h_2}$ sont bloqués dans leur gadget respectif. Ainsi, en supposant que r_{h_2} rentre dans le gadget $Write(Y_j^{c_i})$, d'après la

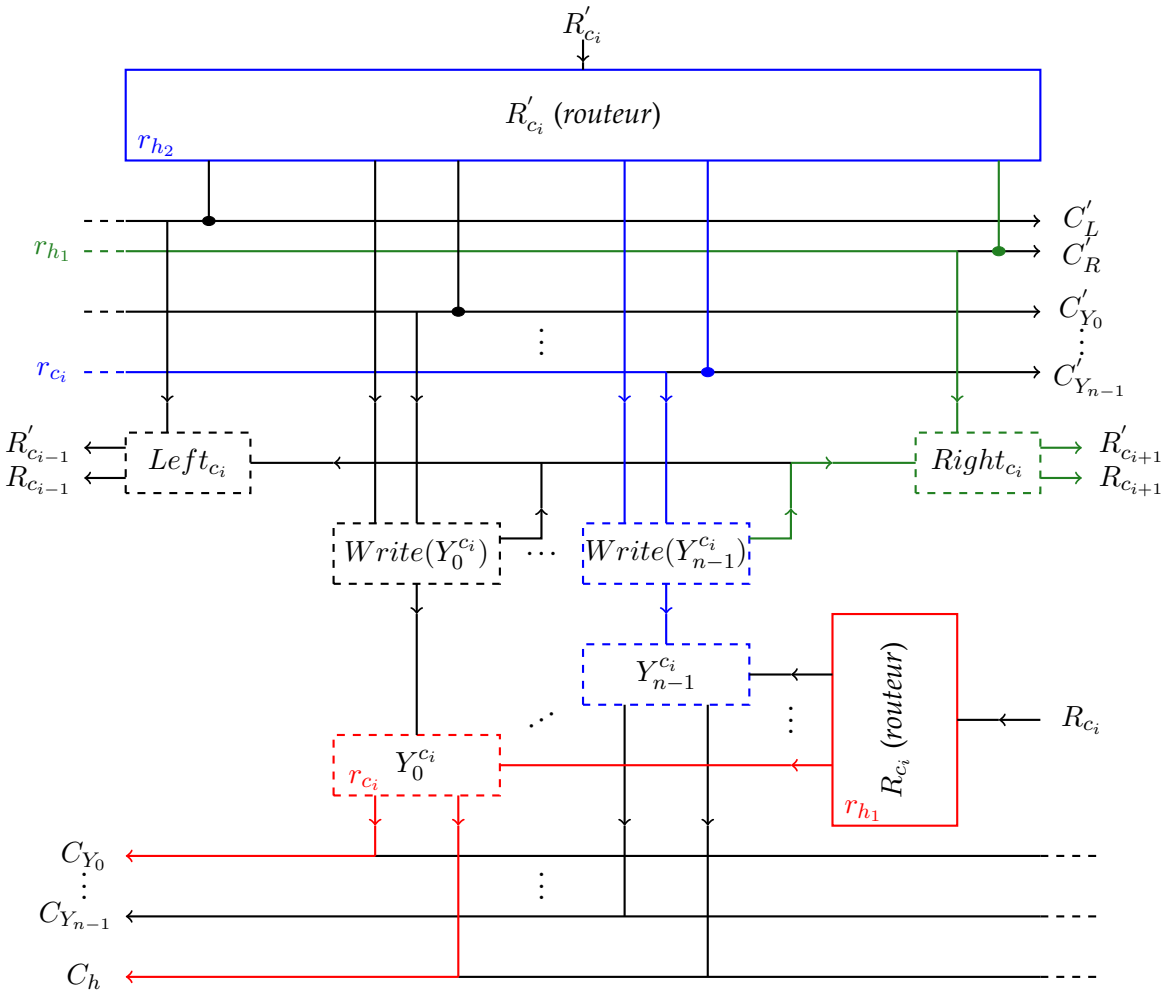


FIGURE 5.21 – *Le gadget* RUBAN *est constitué d'une infinité de gadgets* CELLULE *repartis horizontalement. Cette figure représente le $i^{\text{ème}}$ gadget* CELLULE $Cell_{c_i}$ *du gadget* RUBAN*, il correspond à la $i^{\text{ème}}$ cellule du ruban de M. Les gadgets en pointillés sont des 2-synchronize, voir table 5.1 pour plus de détails sur la représentation des connexions. La sortie $R_{c_{i-1}}$ (resp. $R'_{c_{i-1}}$) est connectée à l'entrée $R_{c_{i-1}}$ (resp. $R'_{c_{i-1}}$) du gadget* CELLULE *directement à gauche dans le gadget* RUBAN *($Cell_{c_{i-1}}$). Les sorties $R_{c_{i+1}}$ (resp. $R'_{c_{i+1}}$) sont connectées à l'entrée $R_{c_{i+1}}$ (resp. $R'_{c_{i+1}}$) du gadget* CELLULE *directement à droite dans le gadget* RUBAN *($Cell_{c_{i+1}}$). En supposant qu'il y ait un robot r_{c_i} dans le gadget $Y_0^{c_i}$ et un robot r_{h_1} dans R_{c_i}. Les deux robots réalisent une opération de lecture en suivant les chemins rouges. Dans cet exemple, le symbole lu est Y_0. Maintenant supposons qu'il y ait un robot r_{h_2} dans le gadget R'_{c_i} et deux robots r_{c_i} et r_{h_1} venant respectivement des canaux de communication $C'_{Y_{n-1}}$ et C'_R. Les trois robots réalisent une opération d'écriture en suivant les chemins bleus et une opération de déplacement en suivant les chemins verts et bleus. Dans cet exemple, l'opération d'écriture réalisée correspond à l'écriture du symbole Y_{n-1} dans la case c_i du ruban du M et l'opération de déplacement à un déplacement de la tête vers la droite, vers la cellule c_{i+1}. Voir lemme 2 pour plus de détails sur ces chemins.*

propriété 4, les robots r_{c_i} et r_{h_2} ont accès aux gadgets $Y_j^{c_i}$ et $Left_{c_i}$ (resp. $Right_{c_i}$), respectivement. Ainsi, r_{h_1} et r_{h_2} accèdent aux gadgets $R_{c_{i-1}}$ et $R'_{c_{i-1}}$ (resp. $R_{c_{i+1}}$ et $R'_{c_{i+1}}$) respectivement.

■

Quand la tête de lecture atteint la $i^{\text{ème}}$ cellule du ruban, (i.e. r_{h_1} et r_{h_2} rentrent dans les gadgets R_{c_i} et R'_{c_i} respectivement) trois opérations sont simulées de la manière suivante.

- **Opération de lecture.** Cette opération est réalisée en envoyant le robot r_{c_i} au gadget CONTRÔLEUR (avec l'aide de r_{h_1}) via le canal de communication C_{Y_j} (lemme 2 (1)). Notez que, après l'opération de lecture, r_{h_1} est envoyé au gadget CONTRÔLEUR via le canal C_h afin d'annoncer la fin de l'opération de lecture (cette action est indispensable et permet au gadget CONTRÔLEUR d'exécuter la fonction de transition d'états. Les gadgets et connexions en rouge sur la figure 5.21 sont un exemple des éléments concernés par l'opération de lecture. Dans cet exemple, r_{c_i} se situe dans le gadget $Y_0^{c_i}$, le robot r_{h_1} le rejoint puis ils sont envoyés au gadget CONTRÔLEUR. Un d'entre eux sort par le canal C_{Y_0} pour signaler la cellule contient le symbole Y_0 et le second par le canal C_h pour informer de la fin de l'opération de lecture.

- **Opération d'écriture.** Le CONTRÔLEUR indique le nouveau symbole Y_k à écrire dans la cellule $Cell_i$ en renvoyant r_{c_i} via le canal de communication C'_{Y_k}. Ainsi, l'opération de lecture est simulée en interceptant le robot r_{c_i} avec r_{h_2} dans le canal C'_{Y_k} afin de l'aider à atteindre le gadget $Y_k^{c_i}$ (après que r_{h_2} l'ait rejoint et aidé à franchir le gadget $Write(Y_k^{c_i})$, voir lemme 2 (2)). Les gadgets et connexions en bleu sur la figure 5.21 sont un exemple des éléments concernés par l'opération d'écriture lorsque le nouveau symbole à écrire est Y_{n-1}. Dans cet exemple, dans un premier temps, r_{h_2} intercepte r_{c_i} dans le canal $C'_{Y_{n-1}}$ afin de l'aider à atteindre le gadget $Write(Y_{n-1}^{c_i})$. Ensuite, r_{h_2} rejoint r_{c_i} dans le gadget $Write(Y_{n-1}^{c_i})$ et l'aide à atteindre le gadget $Y_{n-1}^{c_i}$. Après l'opération d'écriture, le robot r_{h_2} sort du gadget $Write(Y_{n-1}^{c_i})$ et peut rejoindre le gadget $Left_{c_i}$ ou le gadget $Right_{c_i}$.

- **Opération de déplacement.** Le gadget CONTRÔLEUR indique la direction dans laquelle la tête de lecture doit se déplacer en renvoyant le robot r_{h_1} via le canal de communication C'_L ou C'_R (pour gauche ou droite respectivement). Ainsi, l'opération de déplacement à gauche (resp. à droite) est simulée en interceptant le robot r_{h_1} avec r_{h_2} dans le canal C'_L (resp. C'_R) pour lui permettre d'atteindre le gadget $Left_{c_i}$ (resp. $Right_{c_i}$) avant l'opération d'écriture. Ensuite, l'opération d'écriture est réalisée puis r_2 peut rejoindre r_1 dans son gadget et les deux robots sont envoyés dans les gadgets $R_{c_{i-1}}$ et $R'_{c_{i-1}}$, s'il s'agit d'un mouvement vers la gauche (resp. dans les gadgets $R_{c_{i+1}}$ et $R'_{c_{i+1}}$ s'il s'agit d'un mouvement vers la droite). Voir lemme 2 (2). Les gadgets et connexions en vert sur la figure 5.21 sont un exemple des éléments concernés par l'opération de déplacement lorsque la tête doit être déplacée vers la droite.

5.5.3 Contrôleur et fonction de transition d'états

Dans cette sous-section, nous introduisons le gadget CONTRÔLEUR que nous utilisons pour simuler la fonction de transition d'états δ de la machine de Turing M. Le gadget CONTRÔLEUR se décompose en $|Q|$ gadgets ÉTAT (avec Q l'ensemble des états de M).

Construction 5 : [Gadget CONTRÔLEUR] Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ une machine de Turing de m états ($Q = \{q_0, \ldots, q_{m-1}\}$) et n symboles ($\Gamma = \{Y_0, \ldots, Y_{n-1}\}$). Étant donné un gadget COMMUNICATION obtenu par la construction 3, le gadget CONTRÔLEUR (figure 5.22) est construit de la façon suivante.
On commence par créer l'unique case d'arrivée t. Ensuite, pour chaque état $q_i \in Q$, construire un gadget *état* $State_{q_i}$ comme suit :

- construire un gadget $(2n+1)$-routeur (appelé R_{q_i}) ainsi qu'un gadget n-routeur (appelé R'_{q_i}),
- pour chaque symbole $Y_j \in \Gamma$, construire un gadget 3-synchronieur (appelé $\Delta^{q_i}_{Y_j}$).

Pour chaque état $q_i \in Q$ et pour chaque symbole $Y_j \in \Gamma$, soit (p, Y_ℓ, D) la valeur retournée par $\delta(q_i, Y_j)$ (avec $D \in \{L, R\}$). Les gadgets ÉTAT sont connectés comme suit :

- une sortie du gadget R'_{q_i} est connectée à une entrée de $\Delta^{q_i}_{Y_j}$,
- une sortie du gadget R_{q_i} est connectée à une entrée de $\Delta^{q_i}_{Y_j}$,
- le gadget R_{q_i} intersecte le canal C_h vers le gadget R'_{q_i},
- le gadget R_{q_i} intersecte le canal C_{Y_j} vers le gadget $\Delta^{q_i}_{Y_j}$,
- si $\delta(q_i, Y_j)$ n'est pas un cas d'arrêt pour M, alors les trois sorties de $\Delta^{q_i}_{Y_j}$ sont chacune connectées avec une entrée des gadgets R_p, C'_{Y_ℓ} et C'_D,
- si $\delta(q_i, Y_j)$ est un cas d'arrêt pour M, une sortie de $\Delta^{q_i}_{Y_j}$ est connectée à la case d'arrivée t et les deux autres sorties de $\Delta^{q_i}_{Y_j}$ sont fermées avec des murs.

La figure 5.22 illustre le gadget CONTRÔLEUR. Les rôles joués par ce gadget sont les suivants :

- il change l'état de la machine en fonction de l'état actuel et du symbole lu par la tête, et
- il transmet au gadget CELLULE courant le nouveau symbole à écrire et la direction dans laquelle la tête doit se déplacer.

Un robot r_s est utilisé pour encoder l'état courant de M. Le robot r_s se situe dans le gadget R_{q_i} si et seulement si l'état actuel de M est q_i. Par abus de langage, on dit que le gadget CONTRÔLEUR est dans l'état q_i.

Lemme 3 : Considérons le gadget CONTRÔLEUR obtenu par la construction 5 et que le robot r_s est situé dans le gadget R_{q_i} (un des gadgets composant le gadget $State_{q_i}$). Supposons que deux robots r_{c_j} et r_{h_1} viennent respectivement des canaux de communication C_{Y_k} et C_h. Soit (p, Y_ℓ, D) la valeur retournée par $\delta(q_i, Y_j)$ (avec $D \in \{L, R\}$). Alors, les trois robots r_s, r_{c_j} et r_{h_1} ne peuvent sortir du gadget $State_{q_i}$ que par les trois sorties suivantes : R_p, C'_{Y_ℓ} et C'_D, de plus deux robots ne peuvent pas emprunter la même sortie.

Démonstration. D'après la propriété 3, r_s peut intercepter successivement r_{c_j} et r_{h_1} afin de les aider à atteindre respectivement les gadgets $\Delta^{q_i}_{Y_k}$ et R'_{q_i}. Par la construction 5, le robot r_{c_j} ne peut pas accéder à un autre gadget que $\Delta^{q_i}_{Y_k}$. De plus, par la propriété 3, r_{h_1} et r_s peuvent accéder à n'importe quel gadget $\Delta^{q_i}_{Y_{k'}}$ ($\forall Y_{k'} \in \Gamma$). Toutefois, si r_{h_1} ou r_s (ou les deux) se déplacent dans un gadget $\Delta^{q_i}_{Y_{k'}}$ tel que $k' \neq k$, alors, en accord avec la propriété 4, r_s, r_{c_j} et r_{h_1} seront chacun bloqués dans leur gadget respectif. Maintenant, supposons que les robots r_s and r_{h_1} rentrent dans le gadget $\Delta^{q_i}_{Y_k}$. Conformément à la propriété 4, r_s, r_{c_j} et r_{h_1} accèdent chacun à une sortie différente du gadget $\Delta^{q_i}_{Y_k}$. D'après la construction 5, un robot accède au gadget R_p, un second au canal C'_{Y_ℓ} et le dernier au canal C'_D. ■

Étant donné que tous les robots sont de la même couleur dans notre instance, ils sont interchangeables. Ainsi, par soucis de compréhension, on considérera toujours que le robot r_s accède au gadget R_p, r_{c_j} au canal C'_{Y_ℓ} et r_{h_1} au canal C'_D.

Quand le gadget CONTRÔLEUR reçoit les robots r_{h_1} et r_{c_j} venant du gadget RUBAN, il réalise une opération de transition, définie ci dessous.

– **Opération de transition.** Soit (p, Y_ℓ, D) la valeur retournée par $\delta(q_i, Y_j)$ (avec $D \in \{L, R\}$). Soit r_s, un robot situé dans le gadget R_{q_i}, r_{c_j} un robot venant du canal C_{Y_k} et r_{h_1} venant de C_h. Conformément au lemme 3, r_{c_j} et r_{h_1} sont envoyés au gadget RUBAN via les canaux $C'_{Y_{\ell'}}$ et C'_D (avec $D \in \{L, R\}$). Les gadgets et connexions en rouge sur la figure 5.22 sont un exemple des éléments concernés par l'opération de transition lorsque la machine se trouve dans l'état q_0 (le robots r_s est dans le gadget R_{q_0}), que le symbole lu par la tête de lecture est Y_0 (r_{c_i} arrive par le canal C_{Y_0}) et que la fonction de transition $\delta(q_0, Y_0)$ retourne le triplet (q_{m-1}, Y_{n-1}, R).

5.5.4 Construction complète

Dans cette sous-section nous présentons la construction complète de l'instance I du problème d'ACCESSIBILITÉ GÉNÉRALE qui simule une machine de Turing M.

Construction 6 : Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ une machine de Turing de m états ($Q = \{q_0, \ldots, q_{m-1}\}$) et n symboles ($\Gamma = \{Y_0, \ldots, Y_{n-1}\}$). Considérons P un plateau de jeu obtenu par la construction 3, la construction 4 et la construction 5. L'instance $I = (G, R, T)$ du problème d'ACCESSIBILITÉ GÉNÉRALE est créée de la façon suivante. Les robots composant R démarrent dans les positions suivantes :

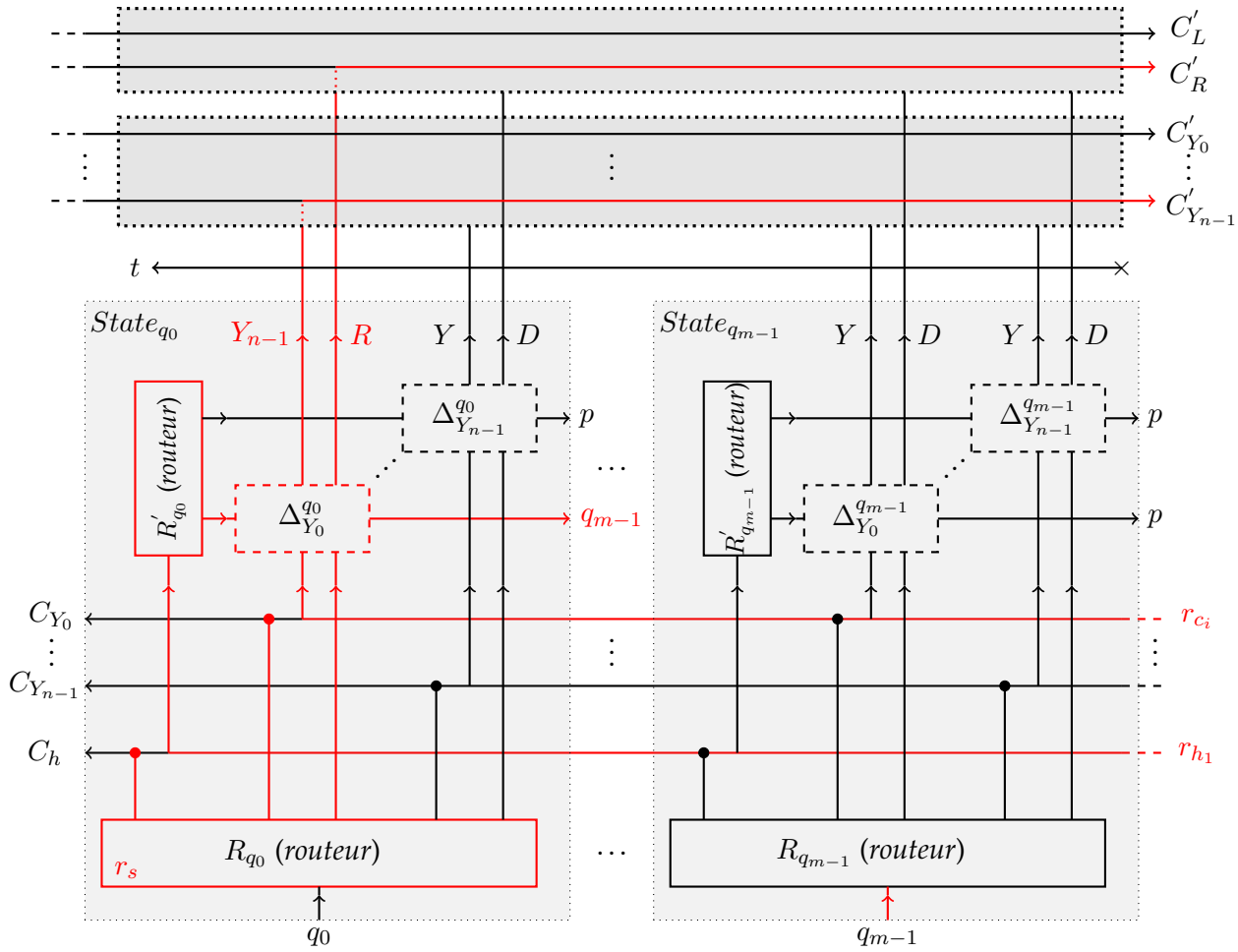


FIGURE 5.22 – *Gadget* CONTRÔLEUR, *gadget contenant l'unique case d'arrivée* t. *Le gadget* CONTRÔLEUR *est décomposé de* $m = |Q|$ *gadgets* ÉTAT *appelés* $State_{q_i}$ *(pour tout* $q_i \in Q$*). Les gadgets en pointillés sont des 2-synchronize, voir table 5.1 pour plus de détails sur la représentation des connexions. La connexion des sorties des gadgets de* $\Delta^{q_i}_{Y_j}$ *dépend de la fonction de transition* δ*. Par exemple, si* $\delta(q_0, Y_0) \to (q_{m-1}, Y_{n-1}, R)$ *alors le gadget* $\Delta^{q_0}_{Y_0}$ *est connecté de la façon suivante : sa sortie* p *est connectée à l'entrée* q_{m-1} *du gadget* $R_{q_{m-1}}$ *(du gadget* $State_{q_{m-1}}$*), sa sortie* Y *est connectée au canal de communication* $C'_{Y_{n-1}}$ *et sa sortie* D *au canal* C'_R*. Supposons qu'il y ait un robot* r_{h_2} *dans le gadget* R_{q_0} *et deux robots* r_{c_i} *et* r_{h_1} *arrivant respectivement des canaux de communication* C_{Y_0} *et* C_h*. Les trois robots réalisent une opération de transition en suivant les chemins rouges (voir lemme 3 pour plus de détails sur l'exécution d'une opération de transition).*

- le robot r_s est placé dans le gadget R_{q_0} (un des gadgets composant le gadget ÉTAT $State_{q_0}$, lui-même composant du gadget CONTRÔLEUR),
- soit c_i la cellule de départ du ruban de M, deux robots r_{h_1} et r_{h_2} sont placés respectivement dans les gadgets R_{c_i} et R'_{c_i} (deux des gadgets composant le gadget $Cell_{c_i}$, lui-même composant du gadget RUBAN),
- pour toute cellule c_i du ruban de M, supposons Y_j le symbole initial de c_i, un robot r_{c_i} est placé dans le gadget $Y_j^{c_i}$ (un des gadgets composant le gadget$Cell_{c_i}$, lui-même composant du gadget RUBAN).

La seule et unique case d'arrivée t est placée dans le gadget CONTRÔLEUR (voir construction 5), et donc $T = t$.

Une *étape* dans l'instance produite par construction 6 est constituée des quatre opérations suivantes : (1) opération de lecture, (2) opération de transition, (3) opération d'écriture et (4) opération de déplacement. lemme 2 et lemme 3 assurent que ces opérations se succèdent les unes après les autres dans I et que aucune autre opération ne soit possible.

Un point important de la preuve repose sur le fait que, même si l'instance obtenue par la construction 6 est de dimension infinie horizontalement, elle peut être encodée dans un langage fini. En effet, il est possible de représenter un plateau de taille infini de façon finie avec un langage proche des expressions régulières pour répéter indéfiniment certains motifs. Soit $I_1 = (P_1, R_1, T_1)$ et $I_2 = (P_2, R_2, T_2)$ deux instances du problème d'ACCESSIBILITÉ GÉNÉRALE avec x_1 la taille horizontale du plateau P_1 (en nombre de cases). L'instance $I_1 \circ I_2$ est créée en juxtaposant I_2 à droite de I_1. Plus précisément, $I_1 \circ I_2 = (P_1 \cup P'_2, R_1 \cup R'_2, T_1 \cup T'_2)$ où (P'_2, R'_2, T'_2) est obtenue par une translation horizontale de x_1 cases appliqué à I_2 (i.e. le plateau, les robots et les cases d'arrivées sont décalés de x_1 cases). Dans notre langage, nous notons I^* une instance correspondant à une infinité d'instances I juxtaposées, autrement dit : $I^* = I \circ I \circ \dots$.

L'instance I obtenue par construction 6 de dimension horizontale infinie peut être encodée dans un langage fini comme suit. Pour tout $\ell \in \Gamma$, C^ℓ désigne un gadget CELLULE $Cell_{c_i}$ avec un robot dans le gadget $Y_l^{c_i}$. Étant donné une entrée $\Sigma = \ell_1 \ell_2 \dots \ell_{|\Sigma|}$ de M, l'ensemble des gadgets CELLULE peuvent être encodés par : "$(C^{Y_0})^* \circ C^{\ell_1} \circ \dots \circ C^{\ell_{|T|}} \circ (C^{Y_0})^*$" (avec Y_0 le symbole correspondant au symbole blanc). Le gadget CONTRÔLEUR est positionné au dessus du premier gadget CELLULE ne contenant pas le symbole blanc (Y_0).

Théorème 8 : Considérant la machine de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, Y_0, F)$ et I l'instance correspondante du problème d'ACCESSIBILITÉ GÉNÉRALE obtenue par construction 6, après k étapes, M et I sont toujours dans le même état. Autrement dit, pour toute entrée Σ et pour tout k, à la $k^{\text{ème}}$ étape dans I et la $k^{\text{ème}}$ transition de M, les conditions suivantes sont respectées :

1. le gadget CONTRÔLEUR est dans l'état q_i si et seulement si M est dans l'état q_i,
2. le gadget CELLULE courant est $Cell_{c_j}$ si et seulement si la tête de M se trouve sur la case c_j du ruban, et
3. pour toute cellule c_j, le gadget CELLULE correspondant $Cell_{c_j}$ contient Y_ℓ (avec $\ell \in \Gamma$) si et seulement si la cellule c_j du ruban de M contient le symbole Y_ℓ.

Démonstration. Par construction, il est clair que (a), (b) et (c) sont vrais pour $k = 0$. Supposons que $k > 0$ étapes ont été réalisées I et M et que les propositions (a), (b) et (c) sont respectées. Soit q_i l'état courant, c_j la case sur laquelle se trouve la tête de lecture, Y_ℓ le symbole contenu dans c_j et $(p, Y_{\ell'}, D)$ (avec $D \in \{L, R\}$) les valeurs retournées par $\delta(q_i, Y_\ell)$.

Dans le paragraphe suivant, nous montrons que les propositions (a), (b) et (c) sont

toujours vérifiées à l'étape $k+1$ dans I et pour la transition $k+1$ dans M.

Comme nous l'avons vu dans le lemme 2 (1), une opération de lecture du gadget CELLULE courant est réalisée lorsque les robots r_{c_j} et r_{h_1} accèdent au gadget CONTRÔLEUR via les canaux de communication C_{Y_ℓ} et C_h.

Ensuite, en accord avec le lemme 3, une opération de transition est réalisée dans le gadget CONTRÔLEUR, le robot r_s (qui représente l'état courant) rentre dans le gadget d'état $State_p$ dans le gadget R_p tandis que r_{c_j} et r_{h_1} sont envoyés vers le gadget RUBAN via les canaux $C'_{Y_{\ell'}}$ et C'_D.

Ensuite, une opération de lecture et une opération de déplacement (lemme 2) sont réalisées dans le gadget RUBAN. Autrement dit, le robot r_{c_j} accède au gadget $Y^j_{\ell'}$, r_{h_1} et r_{h_2} accèdent au gadgets R_{c_j-1} et R'_{c_j-1} (des gadgets composant le gadget $Cell_{c_{j-1}}$) si $D=L$, ou aux gadgets R_{c_j+1} et R'_{c_j+1} (des gadgets composant le gadget $Cell_{c_{j+1}}$) si $D=R$. Maintenant, la machine de Turing M est dans l'état p et sa tête est sur la case c_{j-1} ou c_{j+1} (en fonction de si $D=L$ ou $D=R$).

Ainsi, les propositions (a) et (b) sont vérifiées $k+1$. De plus, la cellule c_j du ruban de M contient maintenant le symbole $Y_{\ell'}$ et le robot r_{c_j} est dans le gadget $Y^j_{\ell'}$ (du gadget $Cell_{c_j}$). Puisque seulement la cellule c_j du ruban de M et le gadget CELLULE $Cell_{c_j}$ ont été modifiés par les opération précédentes, (c) est vérifié pour $k+1$. ■

Théorème 9 : Soit $I=(G,R,T)$ l'instance du problème d'ACCESSIBILITÉ GÉNÉRALE obtenue par construction 6 à partir de la machine de Turing $M=(Q,\Sigma,\Gamma,\delta,q_0,Y_0,F)$ de m états ($Q=\{q_0,\ldots,q_{m-1}\}$) et n symboles ($\Gamma=\{Y_0,\ldots,Y_{n-1}\}$). I admet une configuration gagnante si et seulement si M s'arrête.

Démonstration. Supposons que I admet une configuration gagnante (i.e. un robot accède à la case d'arrivée). Ainsi, les robots ont atteint le gadget $\Delta^{q_i}_{Y_j}$ tel que $\delta(q_i,Y_j)$ est un cas d'arrêt pour M. Donc, en accord avec le théorème 8, les robots ont réalisé une séquence d'opérations de transition telle que la séquence de transitions de M correspondante conduit M de sont état initial à un cas d'arrêt.

Supposons que M atteint un cas d'arrêt $\delta(q_i,Y_j)$. Ainsi, il existe une séquence de transition depuis l'état initial jusqu'à un cas d'arrêt. En accord avec le théorème 8, la séquence d'opérations de transitions correspondante permet aux robots de I d'accéder au gadget $\Delta^{q_i}_{Y_j}$. Un robot peut donc atteindre la case d'arrivée t, alors I admet une configuration gagnante. ■

D'après le théorème 9, n'importe quelle machine de Turing peut être simulée par une instance du problème d'ACCESSIBILITÉ GÉNÉRALE, alors il peut simuler une machine de Turing universelle, prouvant ainsi le théorème 7.

5.6 Conclusion

Dans ce chapitre, nous avons montré que la version optimisation de Ricochet Robots est **Poly-APX**-difficile à partir d'une S-réduction du problème d'ensemble indépendant maximum. Nous avons proposé une autre preuve montrant que la version décisionnelle du jeu est **PSPACE**-complète à partir d'une réduction polynomiale du problème de reconfiguration d'ensembles indépendants. Enfin, nous avons prouvé que la version décisionnelle avec un nombre de robots infini et un plateau infini en montrant qu'une telle instance peut simuler une machine de Turing universelle.

Comme nous l'avons mentionné en début de chapitre, notre intérêt pour le jeu de Ricochet Robots était de l'utiliser pour la preuve de travail (notamment pour le consensus de preuve d'expérience). Les mineurs auraient dû chercher une solution à une instance de Ricochet Robots générée à partir du bloc qu'ils tentent d'ajouter plutôt que de chercher un nonce. Afin d'utiliser le problème correctement, il nous fallait donc être capables de générer des instances aléatoires plus ou moins difficiles (en fonction de la cible locale par exemple), mais après quelques tests et analyses, nous n'avons pas été en mesure d'extraire un vecteur de difficulté viable. De plus, avant de la résoudre, nous ne pouvons pas garantir que l'instance générée est réalisable. En sachant que le problème est **PSPACE**-complet (voir théorème 6) il paraît compliqué d'utiliser un tel problème en tant que preuve de travail.

Conclusion générale

Depuis 2008 et l'utilisation de la blockchain pour des réseaux d'échange de valeurs, de plus en plus d'algorithmes de consensus sont créés. Du consensus dépend entièrement le comportement des nœuds du réseau. Nous avons choisi dans cette thèse d'imaginer et de développer des consensus qui permettront d'inciter les nœuds du réseau à la fidélité et à l'utilisation du réseau. La présente thèse propose deux algorithmes de consensus pour des réseaux d'échanges de valeurs utilisant la blockchain. Ces consensus ont été pensés dans le but d'inciter les utilisateurs du réseau à la participation et à la loyauté respectivement.

Le premier consensus permet d'inciter les utilisateurs d'une blockchain semi-privée à échanger leurs monnaies ou leurs données sur le réseau. Plus un utilisateur utilise le réseau pour échanger, plus sa récompense est importante. Ce consensus est basé sur la preuve d'utilisation que nous avons définie dans le chapitre 3 et permet à des entreprises (telles que des banques ou des assurances) d'échanger des données d'utilisateurs certifiées (avec l'accord de l'utilisateur concerné). Un tel réseau permet aux utilisateurs de reprendre le contrôle sur leurs données personnelles en choisissant de les partager ou non et aux entreprises d'obtenir des données d'utilisateurs certifiées à moindre coût.

Le second consensus propose de modifier le consensus du réseau Bitcoin afin d'inciter les utilisateurs du réseau à la fidélité, à la loyauté. La preuve d'expérience que nous définissons chapitre 4 permet à chaque utilisateur de montrer la quantité d'énergie qu'il a fournie au réseau. Cette preuve nous permet de définir une cible propre à chaque utilisateur en fonction de la quantité de travail qu'il a fournie précédemment. Plus un utilisateur a travaillé pour le réseau, moins le travail qui lui est demandé pour enregistrer un bloc est important. Ainsi, plus un utilisateur est fidèle au réseau, plus il a de chance d'être récompensé. Un grand nombre de fonctions peuvent permettre de calculer le travail à demander à chaque utilisateur et de cette fonction dépend entièrement le comportement des utilisateurs sur le réseau. Il est donc important pour la suite de définir une fonction équilibrée entre inciter à la fidélité tout en offrant la possibilité à de nouveaux utilisateurs d'intégrer le réseau. Une autre amélioration envisageable serait de permettre aux utilisateurs de transférer l'expérience d'un utilisateur à un autre, rendant ainsi monétisable le travail des utilisateurs.

À l'origine, le second consensus devait utiliser une preuve de travail basée sur des instances du jeu Ricochet Robots. Après avoir étudié le problème de décision que présente le jeu de Ricochet Robots, il parait compliqué de choisir la difficulté d'une instance générée et donc impossible de générer efficacement une instance en fonction de la quantité de

travail fourni par un utilisateur.

Idéalement, les prochaines étapes seront de tester nos consensus à plus grande échelle, avec plus de sites, plus de puissance de calcul et plus de temps. Pour ce faire, il nous faudra implémenter nos consensus en prenant en compte les problèmes de sécurité et en y ajoutant les transactions avant de rendre le code public et d'analyser le comportement que les nœuds du réseau adoptent.

Par la suite, nous souhaitons ajouter un système de blockchain fenêtré au consensus de preuve d'expérience en supprimant les blocs plus vieux que l'échantillon de blocs. Ce système permettrait de réduire la taille aux nombre de blocs que l'échantillon de blocs utilise. Toutefois, un tel système présente de nombreux problèmes : comment sauvegarder les transactions antérieures ? Comment est-ce qu'un nouvel entrant recalcule les UTXOs ? Est-ce que les UTXOs des blocs supprimés le sont également ? Si les UTXOs sont supprimées, est-ce qu'elles sont remises en récompense sur les blocs suivants ?

Bibliographie

Références

[1] LM Bach, Branko Mihaljevic, and Mario Zagar. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1545–1550. IEEE, 2018. (Cited on page 57.)

[2] Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences Ii*, pages 329–334. Springer, 1993. (Cited on page 25.)

[3] Cristina Bazgan, Bruno Escoffier, and Vangelis Th Paschos. Completeness in standard and differential approximation classes: Poly-(d) apx-and (d) ptas-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005. (Cited on pages 104 and 110.)

[4] Steven Michael Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. *Colombia University Libraries*, 1992. (Cited on page 22.)

[5] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. proof of activity: Extending bitcoin's proof of work via proof of stake. *ACM SIGMETRICS Performance Evaluation Review*, 2014. (Cited on pages 60 and 63.)

[6] Blockchain.com. Distribution de hashrate bitcoin, 2021. `https://www.blockchain.com/charts/pools`. (Cited on page 57.)

[7] Blockchain.com. Taux de hash total du bitcoin, 2021. `https://www.blockchain.com/charts/hash-rate`. (Cited on pages 54 and 88.)

[8] Nicolas Butko, Katharina A Lehmann, and Veronica Ramenzoni. Ricochet robots—a case study for human complex problem solving. *Proceedings of the Annual Santa Fe Institute Summer School on Complex Systems (CSSS'05)*, 2005. (Cited on pages 95 and 97.)

[9] Changhao Chenli, Boyang Li, Yiyu Shi, and Taeho Jung. Energy-recycling blockchain with proof-of-deep-learning. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 19–23. IEEE, 2019. (Cited on page 76.)

[10] Central Intelligence Agency (CIA). The world fact book, 2016. https://www.cia.gov/the-world-factbook/field/electricity-consumption/country-comparison. (Cited on page 54.)

[11] Foldingcoin company. Foldingcoin whitepaper. Technical report, Foldingcoin, 2018. https://foldingcoin.net/index.php?option=com_content&view=article&id=236&Itemid=653. (Cited on page 76.)

[12] Golem company. Golem whitepaper. Technical report, Golem, 2016. https://golem.network/crowdfunding/Golemwhitepaper.pdf. (Cited on page 76.)

[13] Gridcoin company. Gridcoin whitepaper. Technical report, Gridcoin, 2018. https://gridcoin.us/assets/img/whitepaper.pdf. (Cited on page 76.)

[14] NEM company. Nem, technical reference, version 1.2.1. Technical report, NEM company, 2018. *https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf*. (Cited on page 60.)

[15] Pikcio Company. Pikciochain, the personal data chain, white paper version 2.0, 2018. *https://www.pikcio.com/pikciochain-whitepaper*. (Cited on page 59.)

[16] P. Crescenzi. A short guide to approximation preserving reductions. *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273, 1997. (Cited on page 38.)

[17] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997. (Cited on page 38.)

[18] Erik D Demaine and Martin L Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(1):195–208, 2007. (Cited on page 28.)

[19] Erik D Demaine, Michael Hoffmann, and Markus Holzer. Pushpush-k is pspace-complete. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 159–170. Citeseer, 2004. (Cited on page 97.)

[20] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976. (Cited on page 24.)

[21] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112, 1976. (Cited on page 24.)

[22] Digiconomist. Bitcoin energy consumption index, 2018. https://digiconomist.net/bitcoin-energy-consumption. (Cited on page 54.)

[23] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992. (Cited on pages 28 and 47.)

[24] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985. (Cited on page 24.)

[25] Birgit Engels and Tom Kamphans. Randolphs robot game is np-hard! *Electronic Notes in Discrete Mathematics*, 25:49–53, 2006. (Cited on pages 11, 95, 97 and 99.)

[26] Birgit Engels and Tom Kamphans. Randolph's robot game is np-complete. In *Proceedings of the Twenty-second European Workshop on Computational Geometry (EWCG'06)*, pages 157–160, 2006. (Cited on page 110.)

[27] Martin Gebser, Holger Jost, Roland Kaminski, Philipp Obermeier, Orkunt Sabuncu, Torsten Schaub, and Marius Schneider. Ricochet robots: A transverse asp benchmark. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 348–360. Springer, 2013. (Cited on pages 95 and 97.)

[28] Martin Gebser, Roland Kaminski, Philipp Obermeier, and Torsten Schaub. Ricochet robots reloaded: A case-study in multi-shot asp solving. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation*, pages 17–32. Springer, 2015. (Cited on pages 95 and 97.)

[29] Martin Gill and Geoff Taylor. Preventing money laundering or obstructing business? financial companies' perspectives on "know your customer" procedures. *British Journal of Criminology*, 44(4):582–594, 2004. (Cited on page 64.)

[30] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990. (Cited on page 25.)

[31] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM computing surveys (CSUR)*, 15(4):287–317, 1983. (Cited on page 16.)

[32] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990. (Cited on page 76.)

[33] Adam Hesterberg and Justin Kopinsky. The parameterized complexity of ricochet robots. *Journal of Information Processing*, 25:716–723, 2017. (Cited on page 97.)

[34] Jaap-Henk Hoepman. Distributed double spending prevention. In *International Workshop on Security Protocols*, pages 152–165. Springer, 2007. (Cited on page 17.)

[35] Markus Holzer and Stefan Schwoon. Assembling molecules in atomix is hard. *Theoretical computer science*, 313(3):447–462, 2004. (Cited on pages 97, 99 and 110.)

[36] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008. (Cited on page 32.)

[37] Falk Hüffner, Stefan Edelkamp, Henning Fernau, and Rolf Niedermeier. Finding optimal solutions to atomix. In *Annual Conference on Artificial Intelligence*, pages 229–243. Springer, 2001. (Cited on pages 97 and 121.)

[38] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. Exploring an unknown cellular environment. In *EuroCG*, pages 140–143, 2000. (Cited on page 97.)

[39] Christian Icking, Tom Kamphans, Rolf Klein, and Elmar Langetepe. Exploring simple grid polygons. In *International Computing and Combinatorics Conference*, pages 524–533. Springer, 2005. (Cited on page 97.)

[40] Takehiro Ito, Erik D Demaine, Nicholas JA Harvey, Christos H Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. (Cited on page 110.)

[41] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999. (Cited on pages 28 and 47.)

[42] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007. (Cited on page 76.)

[43] Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical computer science*, 439:9–15, 2012. (Cited on page 110.)

[44] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper*, 2012. (Cited on pages 29, 59, 60 and 76.)

[45] Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 254–266. IEEE, 1977. (Cited on pages 99 and 110.)

[46] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. (Cited on pages 17, 18 and 19.)

[47] Jorick Lartigau, Fabien Bucamp, and Didier Collin de Casaubon. Pikciochain: a new eco-system for personal data, 2018. (Cited on page 59.)

[48] S. Masseport, B. Darties, R. Giroudeau, and J. Lartigau. Proof of experience: empowering proof of work protocol with miner previous work. In *2020 2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 57–58, 2020. (Cited on pages 12 and 94.)

[49] Samuel Masseport, Jorick Lartigau, Benoit Darties, and Rodolphe Giroudeau. Proof of usage: User-centric consensus for data provision and exchange. In *BRAINS: Blockchain, Robotics and AI for Networking Security*, 2019. (Cited on pages 12 and 74.)

[50] Samuel Masseport, Jorick Lartigau, Benoit Darties, and Rodolphe Giroudeau. Proof of usage: user-centric consensus for data provision and exchange. *Annals of Telecommunications*, 75(3):153–162, 2020. (Cited on pages 12 and 74.)

[51] Earl H McKinney. Generalized birthday problem. *The American Mathematical Monthly*, 73(4):385–387, 1966. (Cited on page 21.)

[52] Ralph Merkle and Martin Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE transactions on Information Theory*, 24(5):525–530, 1978. (Cited on page 24.)

[53] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987. (Cited on page 23.)

[54] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013. (Cited on page 60.)

[55] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE, 2014. (Cited on page 76.)

[56] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985. (Cited on pages 24, 45 and 77.)

[57] Wouter H Muller, Christian H Kalin, and John G Goldsworth. *Anti-Money Laundering: international law and practice*. John Wiley & Sons, 2007. (Cited on page 64.)

[58] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. (Cited on pages 9, 25, 39, 43, 44 and 59.)

[59] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Annual International Cryptology Conference*, pages 617–630. Springer, 2003. (Cited on pages 22 and 23.)

[60] University of Cambrige Judge Business School. Cambridge bitcoin electricity consumption index, 2021. https://cbeci.org/cbeci/comparisons. (Cited on page 54.)

[61] Ivan Osipkov, Eugene Y Vasserman, Nicholas Hopper, and Yongdae Kim. Combating double-spending using cooperative p2p systems. In *27th International Conference on Distributed Computing Systems (ICDCS'07)*, pages 41–41. IEEE, 2007. (Cited on page 17.)

[62] General Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46. *Official Journal of the European Union (OJ)*, 2016. *https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL*. (Cited on pages 59, 61 and 63.)

[63] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. (Cited on page 24.)

[64] JP Robinson. Optimum golomb rulers. *IEEE Transactions on Computers*, 12(C-28):943–944, 1979. (Cited on page 86.)

[65] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014. (Cited on page 54.)

[66] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014. (Cited on page 60.)

[67] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. (Cited on pages 34 and 121.)

[68] James B Shearer. Some new optimum golomb rulers. *IEEE Transactions on Information Theory*, 36(1):183–184, 1990. (Cited on page 86.)

[69] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937. (Cited on page 32.)

[70] Qian Wang, Timothy Dunlap, Youngho Cho, and Gang Qu. Dos attacks and countermeasures on network devices. In *2017 26th Wireless and Optical Communication Conference (WOCC)*, pages 1–6, 04 2017. (Cited on page 72.)

[71] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 557–564. IEEE, 2017. (Cited on page 25.)