Algorithmes efficaces et complexité

Richard Lassaigne
IMJ/Logique mathématique
CNRS-Université Paris Diderot

L'objet de cet article est d'illustrer la notion d'algorithme efficace et de montrer que la théorie de la complexité permet de classifier les problèmes suivant leur difficulté à l'aide de mesures telles que le temps de calcul et l'espace mémoire nécessaires à leur résolution sur ordinateur. Les principales questions abordées sont :

- qu'est-ce qu'un algorithme *efficace* ?
- existe-t-il des problèmes *intrinsèquement difficiles*, c'est-à-dire pour lesquels on ne connaît pas d'algorithme efficace de résolution ?
- comment mesurer la *complexité* d'un problème et comparer deux problèmes suivant ce critère ?
- existe-t-il des méthodes permettant de réduire la complexité d'un problème ?
- l'existence de problèmes intrinsèquement difficiles peut-elle être un atout ?

Une question préalable et fort intéressante, mais non traitée dans cet article, est celle de la modélisation de la *calculabilité*, en particulier de la notion même d'*algorithme*. Parmi les plus célèbres contributions, depuis le début des années 1930, on peut mentionner les approches d'A. Turing, A. Church, S.C. Kleene et K. Gödel. L'intérêt de ces diverses définitions de la calculabilité est qu'elles se sont révélées être toutes équivalentes et qu'elles ont permis de mettre en évidence les limites : l'existence de fonctions non calculables et de problèmes indécidables. Le lecteur intéressé pourra par exemple consulter la référence [3].

Certains problèmes rencontrés quotidiennement utilisent des algorithmes reconnus comme étant très efficaces :

- l'interrogation d'une base de données (train, avion, voyages,...)
- la localisation à l'aide d'un GPS,
- la recherche du plus court chemin entre deux destinations,
- la recherche d'informations à l'aide d'un moteur de recherche.

La première partie de l'article aborde la théorie de la complexité [3, 5] qui repose sur les deux mesures classiques que sont le *temps* de calcul et l'*espace* mémoire nécessaires à la résolution d'un problème donné. Elle permet de faire la distinction entre problèmes

faciles, c'est-à-dire pour lesquels on connaît des algorithmes de résolution efficaces, et des problèmes réputés difficiles. L'introduction de la notion de *réduction* entre problèmes a pour conséquence une caractérisation de certains problèmes représentatifs de cette classe de difficulté. La conjecture $P = NP$, énoncée de manière implicite par Kurt Gödel en 1957 et de manière explicite par Stephen Cook en 1971, pose la question de savoir s'il existe un espoir raisonnable de trouver des algorithmes efficaces pour cette classe de problèmes difficiles.

La deuxième partie s'intéresse à l'exemple de la recherche d'informations à l'aide d'un moteur et montre qu'un problème, qui peut sembler difficile à priori, puisqu'il met en jeu comme dernière étape le calcul du rang de classement d'une page web, peut être résolu à l'aide d'algorithmes efficaces qui font appel à des méthodes mathématiques fort intéressantes. Cette section est inspirée de l'exposé de M. Habib [4].

La dernière partie utilise de manière essentielle la notion de réduction pour montrer comment l'existence d'un problème réputé difficile comme, par exemple, la factorisation des nombres entiers, est un avantage pour les *preuves de sécurité* de protocoles cryptographiques. La section sur la sécurité prouvable est présentée suivant les idées de D. Pointcheval [6].

1 Les classes de complexité P et NP. La réduction : un outil de comparaison des problèmes

L'idée de base de la complexité de calcul des algorithmes a été introduite par J. Hartmanis et R. Stearns en 1965. Les mesures de complexité, qui sont devenues classiques, sont le temps et l'espace nécessaires à la réalisation d'un calcul, ou à la décision d'une propriété. Elles reposent sur l'évaluation du nombre d'opérations *élémentaires* mises en oeuvre . Par exemple, la multiplication de deux entiers de taille m et n nécessite $m \times n$ petites multiplications et additions. Si l'on considère un problème comme celui de la multiplication de deux matrices A et B carrées d'ordre n, le nombre d'opérations élémentaires nécessaires pour obtenir la matrice produit $C = A \times B$ est :

- pour chaque élément de la matrice produit, n multiplications et $n - 1$ additions,
- au total, $n^2 \times (2n - 1) = O(n^3)$ opérations élémentaires.

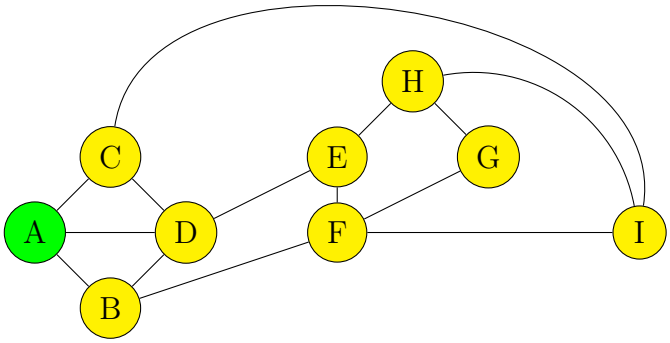
La résolution de ce problème met ainsi en jeu un nombre polynomial d'étapes.

Dans la suite, on s'intéresse aux problèmes de décision pour illustrer la définition des classes de complexité P et NP, mais on pourrait définir de manière analogue des classes de

complexité pour les problèmes de calcul. Les deux classes considérées sont les suivantes :

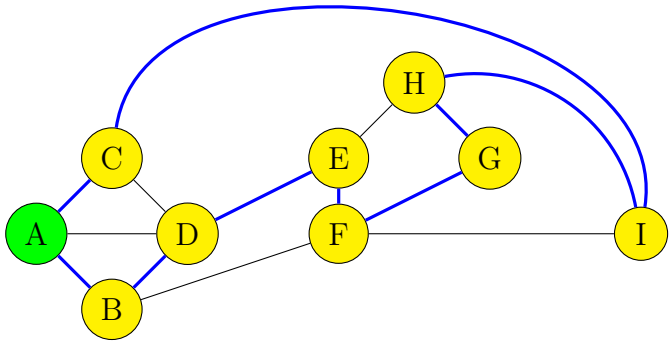
- P est la classe des problèmes *faciles à décider*, c'est-à-dire de complexité en temps de calcul raisonnable,
- NP est la classe des problèmes *faciles à vérifier*, mais difficiles à décider dans l'état actuel de nos connaissances.

Pour donner un exemple de problème de la classe NP, on considère le problème du graphe hamiltonien : étant donné un graphe $G = (E, R)$, c'est-à-dire un ensemble E de sommets et un ensemble R d'arêtes entre ces sommets, et un sommet $s \in E$, peut-on trouver un circuit empruntant des arêtes du graphe, commençant et finissant en s, et passant une fois et une seule par tout autre sommet ? Un exemple de graphe est donné dans la figure suivante. La résolution de ce problème est aussi un pré-requis pour celle du problème du voyageur de commerce : la donnée est alors un graphe valué, c'est-à-dire avec un coût associé à chaque arête, un sommet et une borne entière. La question posée est l'existence d'un circuit parcourant tous les sommets une fois et une seule à partir du sommet considéré et dont le coût total de parcours soit inférieur à la borne donnée.



Graphe avec origine donnée

Pour mesurer la complexité d'un problème sur les graphes, une opération "élémentaire" est la prise en compte d'un sommet ou d'une arête et le nombre d'étapes est évalué en fonction de la taille du graphe, qui est égale à la somme du nombre de sommets et du nombre d'arêtes. En ce qui concerne le problème du graphe hamiltonien, illustré dans la figure suivante, on peut être amené à essayer tous les chemins possibles à partir d'un sommet, ce qui nécessite : $(n-1)(n-2)\ldots 2 = (n-1)!$ étapes avec $n =$ nombre de sommets. Le nombre d'étapes est *exponentiel* dans la taille du graphe. Ce problème est considéré comme difficile à résoudre.



Circuit hamiltonien d'origine donnée

La classe P est définie comme la classe des problèmes de décision pour lesquels il existe un algorithme de décision fonctionnant en *temps polynomial*. En 1964, A. Cobham montre que la classe des problèmes décidables en temps polynomial est indépendante d'un modèle particulier de machine déterministe. et que de nombreuses fonctions mathématiques sont calculables en temps polynomial. En 1965, J. Edmonds affirme que cette classe est un bon modèle pour la classe des problèmes pouvant être résolus de manière efficace. Il remarque également le grand nombre de problèmes connus décidables en temps polynomial et que cette classe reste la même suivant de nombreux et différents modèles raisonnables de calcul. Dans un autre article publié la même année, il donne une description informelle de la classe NP. Concernant la classe P, il faut remarquer que, si le degré du polynôme bornant le temps de calcul est élevé, l'algorithme considéré n'est pas très efficace... D'autre part, certains algorithmes fonctionnant en temps polynomial ont été très durs à trouver et à comprendre.

Le problème du graphe hamiltonien possède la propriété suivante : on peut facilement, c'est-à-dire en temps polynomial, vérifier qu'un parcours dans le graphe passe par tous les sommets une fois et une seule. Cette situation est assez fréquente. Par exemple le problème du coloriage d'un graphe est défini de la manière suivante : étant donné un graphe et un entier k, existe-t-il un *bon* coloriage des sommets, c'est-à-dire un coloriage à l'aide de k couleurs, tel que pour chaque couple de sommets réunis par une arête, les couleurs des deux sommets soient différentes ? Si l'on se donne un graphe, un entier et un coloriage, il est facile de vérifier si ce coloriage est bon. Cependant, tous les problèmes difficiles ne possèdent pas forcément cette propriété : par exemple, déterminer l'existence d'une stratégie gagnante dans certains jeux classiques comme les échecs ou le jeu de Go.

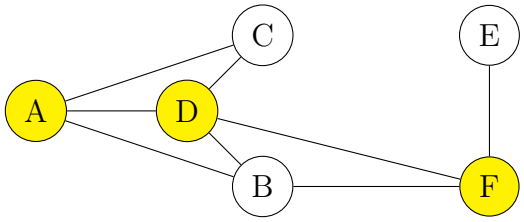
La classe NP est la classe des problèmes *vérifiables* en temps polynomial. Un problème de décision A est vérifiable en temps polynomial si pour toute instance x de ce problème :

- si $x \in A$, il existe un *témoin* y, de taille polynomiale en la taille de x, permettant de décider en temps polynomial que $x \in A$,
- si $x \notin A$, il n'existe pas de tel témoin.

L'importance de la classe de complexité en temps polynomial sur machine non déterministe, d'où le nom NP, a été illustrée par S. Cook (1971), L. Levin (1973) ainsi que R. Karp (1972). La conjecture $P = NP$ peut s'énoncer sous la forme : les problèmes vérifiables en temps polynomial sont-ils aussi décidables en temps polynomial ? Si l'on préfère la forme négative $P \neq NP$: existe-t-il des problèmes dont les solutions sont faciles à vérifier mais difficiles à trouver ?

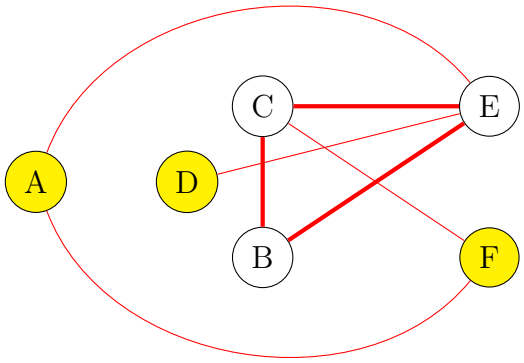
A la fin des années 60, une classe très large de problèmes intéressants sur le plan des applications fut reconnue comme résistant aux nombreuses tentatives pour montrer qu'ils possédaient des algorithmes fonctionnant en temps polynomial. Ces problèmes sont de type combinatoire ou d'optimisation, comme le voyageur de commerce ou certains problèmes de programmation linéaire. Ils possèdent tous un très grand nombre de solutions possibles, sans qu'il y ait de manière facile pour trouver une solution optimale autrement que par recherche du type *force brute*. Comme beaucoup de temps et d'efforts ont été dépensés sans succès pour tenter de résoudre efficacement ces problèmes, l'idée se fit jour qu'il n'existait peut-être pas de telle solution et que ces problèmes étaient en un certain sens difficiles pour les mêmes raisons.

Les contributions fondamentales de S. Cook, L. Levin et R. Karp ont été de prouver que tous ces problèmes de la classe NP sont représentatifs de la difficulté de cette classe : ils sont dits NP-*complets*. S. Cook a montré qu'un problème de base de la logique, noté SAT, est de ce type : étant donné un énoncé de la logique propositionnelle, sous forme normale conjonctive, décider s'il existe une valuation le satisfaisant. L'outil principal de la preuve est la construction d'une *réduction*, c'est-à-dire un algorithme fonctionnant en temps polynomial, permettant de ramener la résolution de n'importe quel problème de la classe NP à la résolution du problème SAT. La contribution essentielle de R. Karp fut de montrer l'existence de réductions du problèmes SAT aux différents problèmes combinatoires ou d'optimisation précédents. Ainsi, un problème de la classe NP est NP-complet si tout problème de la classe se réduit à ce problème donné : il est au moins aussi difficile que tout autre problème dans NP.



Graphe avec recouvrement de sommets minimal

Afin de donner un exemple facile de réduction, on va considérer deux problèmes sur les graphes. Le problème de recouvrement de sommets minimal consiste à déterminer si dans un graphe $G = (E, R)$ donné, il existe un sous-ensemble de sommets S, de taille minimale, tel que toute arête de G a un de ses sommets dans S, comme dans le cas de la figure précédente. Le problème du sous-garphe complet maximal est de déterminer si dans un graphe $G' = (E', R')$, il existe un sous-ensemble de sommets T, de taille maximale, tel que tout couple de sommets de T est lié par une arête du graphe. Une réduction du premier problème au second est la construction, en temps polynomial par rapport à la taille du graphe G, d'un autre graphe G' tel que : le graphe G a un recouvrement de taille minimale si et seulement si le graphe G' a un sous-graphe complet de taille maximale. En fait, cette construction est facile comme le montre la figure suivante : il suffit de considérer, à partir du graphe $G = (E, R)$, le graphe G' obtenu sur le même ensemble de sommets E avec la relation $R' = E^2 - R$ complémentaire de la relation R. Le premier problème est donc au moins aussi difficile que le second.



Graphe avec sous-graphe complet maximal

Dans un article de 1972, qui a eu une influence énorme, Richard Karp, a montré qu'une vingtaine de problèmes importants de combinatoire ou d'optimisation sont tous NP-complets. Cet article présente plusieurs méthodes clés pour prouver la NP-complétude à l'aide de réductions à partir de problèmes déjà connus pour être NP-complets. Il donne un cadre général pour prouver des résultats de NP-complétude ainsi que plusieurs techniques utiles pour établir de telles preuves. Dans les années suivantes, et jusqu'à aujourd'hui, des milliers de problèmes ont été montrés NP-complets [2], confortant ainsi l'affirmation de R. Karp suivant laquelle : « *computational intractability is the rule rather the exception* ». Prouver qu'un problème de la classe NP est NP-complet nous dit qu'il est aussi difficile à résoudre que n'importe quel autre problème de cette classe. En d'autres termes, s'il existe un problème NP-complet qui possède un algorithme efficace, alors c'est également le cas pour tout autre problème de cette classe. Cette question qui est une autre formulation du problème $P = NP$ a résisté aux efforts des chercheurs depuis 1970 et est reconnue comme l'un des principaux problèmes ouverts des mathématiques. Le problème $P = NP$ a été présenté par le Clay Mathematics Institute, à l'occasion du millénium, comme l'un des sept problèmes mathématiques faisant chacun l'objet d'un prix d'un milllion de dollars.

En face de la difficulté reconnue d'un problème NP-complet, les chercheurs se sont tournés vers d'autres méthodes de résolution :

1. la recherche d'*heuristiques* pour résoudre certaines instances du problème, car toutes les instances sont loin d'être uniformément difficiles,
2. une méthode d'*approximation* permettant de trouver une solution proche de l'optimum,
3. un algorithme *probabiliste* permettant de trouver une solution avec une très grande probabilité,
4. une méthode d'*approximation probabiliste* permettant de trouver une solution proche de l'optimum avec une très grande probabilité.

Dans le cas de l'exemple du calcul du Page Rank, c'est une approche probabiliste couplée avec une méthode d'approximation qui permet d'obtenir un algorithme efficace.

2 Un exemple d'algorithme efficace : le calcul du rang de classement des pages dans un moteur de recherche

Le fonctionnement d'un moteur de recherche peut être schématisé comme une suite de trois procédures algorithmiques qui prend en entrée une question représentée par une

chaîne de caractères et fournit comme résultat une liste ordonnée d'url (*uniform resource location*, c'est-à-dire une adresse IP avec un chemin d'accès) :

1. l'extraction du contenu de la question sous forme de quelques mots-clés,
2. la recherche des pages web qui contiennent ces mots-clés,
3. le tri et l'affichage d'une liste ordonnée d'url.

Cette recherche peut être représentée théoriquement comme la recherche dans un graphe gigantesque. Elle donne lieu à la constitution d'une base de données elle aussi gigantesque, avec des moyens de calcul énormes. Elle utilise des algorithmes efficaces, en particulier dans la dernière étape qui consiste à obtenir le classement des pages à l'aide d'un coefficient nommé Page Rank (PR). Dans la suite, on s'intéresse uniquement à ce dernier algorithme.

Le principe de l'algorithme, dû à S. Brin, L. Page, R. Motwani et T. Winograd repose sur l'idée suivante : éviter le plus possible de considérer le contenu sémantique d'une page. L'importance d'une page ne dépend que de la structure des liens entre les pages html. Ceci aboutit à une formulation récursive : une page aura un score d'autant plus élevé qu'elle est référencée par des pages ayant un score élevé. On peut remarquer que le Page Rank n'est sans doute plus utilisé dans sa forme d'origine, mais l'idée de base est toujours ce qui est le plus important dans le moteur.

Le *graphe* du web est un graphe orienté dont les sommets sont les *pages* html et dont les arcs correspondent aux *liens* entre les pages. L'exploration de ce graphe est un problème difficile d'informatique distribuée : des programmes appelés robots suivent les liens. La propriété intéressante est qu'une page standard contient au plus une centaine de liens vers d'autres pages. Le *degré sortant* d'un sommet est donc borné par une quantité tout-à-fait raisonnable et le graphe, qui est peu dense, est représentable efficacement. L'exploration de nouvelles pages se fait en approximativement un mois. Périodiquement, la base de données est mise à jour et le classement recalculé sur l'image du graphe du mois précédent.

Le Page Rank a été pensé comme une mesure de la popularité des pages web. Le modèle proposé par S. Brin et L. Page est celui d'un surfeur aléatoire qui se promène le long de ce graphe orienté : si à l'étape n, il se trouve sur le sommet i, alors à l'étape suivante $n+1$, il se déplace de manière aléatoire uniforme jusqu'à l'un des sommets j reliés à i. Le parcours du graphe peut ainsi être représenté par une matrice A dont le coefficient $A(i,j)$ peut être considéré comme la probabilité que le surfeur passe de la page i à la page j en une unité de temps. L'hypothèse faite est celle d'une répartition équitable entre les liens sortant d'une page : $A(i,j) = \frac{1}{deg(i)}$ où $deg(i)$ est le degré sortant du sommet

i, c'est-à-dire le nombre de liens issus de cette page, s'il existe un arc allant de i à j et $A(i,j) = 0$ sinon. On remarque que, dans le cas où il existe des liens sortant d'une page i, la répartition équitable entre ces liens a pour conséquence que la somme des éléments de la ligne i est égale à 1 : $\sum_j A(i,j) = 1$. La transformation associée à cette matrice peut s'interpréter comme le passage d'un état à un autre dans une *chaîne de Markov*. Le calcul du coefficient associé à chaque page i est effectué comme la limite d'une suite $R_n(i)$ qui est la probabilité de se trouver sur la page i à l'étape n du calcul :

$$R_{n+1}(i) = \sum_{j \to i} \frac{1}{deg(j)} R_n(j)$$

où la somme est réalisée sur tous les arcs (j,i). On obtient ainsi une formulation vectorielle de ces équations dans laquelle R_n représente le vecteur des coefficients des différentes pages à l'étape n du calcul : $R_{n+1} = A^t R_n$ où A^t est la matrice transposée de la matrice A. Le vecteur final $R = lim_{n \to \infty} R_n$, si cette limite existe, est un point fixe de la transformation et correspond à une distribution *stationnaire* de la chaîne de Markov associée au parcours du graphe. De plus, si le graphe est fortement connexe, c'est-à-dire si tout sommet est accessible à partir d'un autre sommet, la matrice A est une matrice dite *stochastique* et la distribution stationnaire existe et est unique.

Malheureusement, le graphe du web n'est en général pas fortement connexe. Pour pallier à cet inconvénient, le comportement du surfeur est modifié de la manière suivante. On initialise d'abord à $1/N$ le coefficient PR de toutes les pages, où N est le nombre total de pages du graphe. Le surfeur peut choisir à partir d'une certaine page :

- soit de zapper sur une page aléatoire avec probabilité $1 - \alpha$
- soit de suivre un lien sortant avec probabilité α

Le calcul du coefficient de chaque page est ainsi modifié :

$$R_{n+1}(i) = \frac{(1-\alpha)}{N} + \alpha \sum_{j \to i} \frac{1}{deg(j)} R_n(j)$$

Le graphe sous-jacent à cette marche aléatoire devient fortement connexe et on considère la transformation associée $T : \mathbb{R}^N \longrightarrow \mathbb{R}^N$ définie par :

$$T(x) = (1-\alpha)e + \alpha A^t x$$

où e est le vecteur dont toutes les composantes valent$1/N$. Si la matrice A est stochastique, alors l'application T est contractante de rapport α et quel que soit le vecteur initial x_0,

la suite $(x_n)_{n\in\mathbb{N}}$ définie par $x_{n+1} = T(x_n)$ converge vers un unique point fixe μ. La vitesse de convergence est régie par :

$$\mid x_{n+1} - \mu \mid \leq \frac{\alpha}{(1-\alpha)} \mid x_{n+1} - x_n \mid$$

avec $\mid y \mid = \sum_i \mid y_i \mid$. Pour obtenir un test d'arrêt, il suffit de choisir une borne d'*approximation* ε et de tester si $\mid x_{n+1} - x_n \mid \leq \frac{(1-\alpha)}{\alpha}\varepsilon$. Une estimation grossière du nombre d'itérations nécessaires pour obtenir une approximation à ε près est $log(\varepsilon)/log(\alpha)$ [1] ce qui donne une centaine d'itérations pour $\varepsilon = 10^{-8}$ et $\alpha = 0.85$.

Les raisons pour lesquelles le calcul du coefficient PR est tant utilisé sont les suivantes :

- la *convergence* est très rapide ;
- le calcul peut s'effectuer simplement de manière parallèle ;
- il possède plusieurs interprétations mathématiques intéressantes ;
- il est extrêmement *robuste* expérimentalement car peu dépendant des conditions initiales.

En résumé, les principaux avantages du calcul du coefficient PR sont que le calcul s'effectue hors ligne et qu'il est robuste. Les principaux inconvénients sont que le PR favorise les pages recevant beaucoup de liens et que les thématiques marginales ne peuvent ressortir. Deux approches ont été proposées pour pallier à ces problèmes : le surf intelligent et le page rank *thématique*. L'idée du surf intelligent consiste à modifier le comportement du surfeur aléatoire pour le rendre sensible à la sémantique. Pour le réaliser, il est possible, théoriquement, de sauter de page en page de manière probabiliste mais avec des probabilités conditionnées par le contenu des pages et des requêtes de l'utilisateur. Le problème est que la mise en oeuvre nécessite une puissance de calcul hors de portée. L'idée de la seconde approche est d'associer à chaque page non plus un coefficient, mais un vecteur de coefficients PR, chaque composante correspondant à l'un des sujets évoqués dans la requête de l'utilisateur.

Le fonctionnement d'un moteur de recherche peut ainsi se décomposer de manière un peu plus précise dans les étapes suivantes :

1. le précalcul d'un fichier inversé des pages web dans une gigantesque base de données distribuée,
2. l'extraction du contenu de la question (mots clés),
3. la recherche de toutes les pages web contenant ces mots clés et le calcul d'un score pondéré pour chaque page,
4. le filtrage des pages résultats à l'aide d'un profil utilisateur (langue, adresse IP, académique versus commercial),

5. le tri des pages obtenues à l'aide du calcul du Page Rank (plus des astuces secrètes) et l'affichage de cette liste ordonnée.

Du point de vue de la complexité, le fonctionnement d'un moteur de recherche repose sur des algorithmes très efficaces grâce à l'utilisation d'algorithmes *probabilistes* et de méthodes d'*approximation*.

3 Complexité et preuves de sécurité des protocoles cryptographiques

Les objectifs de la cryptographie peuvent être définis en terme de propriétés que les protocoles doivent assurer :

- la confidentialité, c'est-à-dire le contenu d'un message est caché,
- l'authenticité, l'auteur d'un message est bien identifié,
- l'intégrité, les messages n'ont pas été altérés.

Deux approches peuvent être considérées pour les preuves de sécurité suivant que l'on se place dans un cadre général de secret parfait ou de secret en pratique. Dans le premier cas, aucune information ne doit pouvoir être extraite du message crypté, même par un adversaire ayant des capacités illimitées en temps et puissance de calcul. Cette approche relève de la *théorie de l'information*. Dans le second cas, on considère qu'en pratique, les participants au protocole, légitimes et adversaires, sont limités en temps et en puissance de calcul. L'approche qui nous intéresse ici relève de la *théorie de la complexité*.

La *sécurité prouvable* tente de répondre à la question : qu'est-ce qu'un schéma cryptographique sûr ? Dans le cadre, par exemple de la cryptographie asymétrique, c'est-à-dire à clé publique, l'objectif est de montrer que le secret de la clé privée garantit le secret des communications. Pour cela, il est nécessaire de définir formellement les propriétés de sécurité et de montrer que le schéma cryptographique les satisfait. La sécurité prouvable utilise la réduction, outil essentiel de la théorie de la complexité, pour montrer que casser un schéma cryptographique est au moins aussi dur que de résoudre de manière efficace un problème reconnu comme étant difficile. La preuve par l'absurde consiste à faire l'hypothèse qu'il existerait un adversaire capable de mettre au point une attaque permettant de casser le schéma cryptographique considéré. La construction de la réduction s'effectue en utilisant cette attaque comme sous-programme. Cette réduction permettrait alors de résoudre de manière efficace un problème réputé difficile, ce qui contredit le fait qu'il n'existe pas d'algorithme efficace de résolution de ce problème.

Pour prouver la sécurité d'un schéma cryptographique, on a ainsi besoin :

- d'un modèle formel des propriétés de sécurité,
- de la notion de réduction,
- d'hypothèses de complexité, comme l'existence de problèmes reconnus comme difficiles,
- d'un modèle de participants, légitimes et adversaire, avec limitation de la puissance de calcul.

La modélisation est faite sous la forme d'un *jeu de sécurité* entre l'adversaire et un challenger qui lui pose des questions. L'adversaire a accès à des *oracles*, qui lui permettent d'obtenir des informations extérieures concernant par exemple le déchiffrement ou la signature. Le challenger pose des questions et à la fin de ce jeu déclare si l'adversaire a gagné. Ce jeu de sécurité modélise le jeu que l'on souhaite que l'adversaire ne soit pas en mesure de gagner en temps raisonnable avec une probabilité trop importante. Ensuite, la construction de la réduction revient à *simuler* les oracles et le challenger en interaction avec l'adversaire. Ce simulateur prend alors en entrée une instance d'un problème difficile et exploite la puissance de calcul de l'adversaire pour résoudre de manière efficace cette instance. La puissance de calcul considérée est celle d'un algorithme *probabiliste* fonctionnant en *temps polynomial*.

Il reste maintenant à identifier les problèmes que l'on reconnaît comme difficiles et à préciser les bornes de complexité que l'on s'autorise, ainsi que le comportement des réductions vis-à-vis de ces bornes. Le plus classique des problèmes difficiles est celui de la *factorisation* d'un nombre entier en nombres premiers. Le système à clé publique RSA dû à R. Rivest, A. Shamir et L. Adleman en 1977 est fondé sur le fait qu'il est facile de multiplier des nombres entiers, mais qu'il est difficile de factoriser de grands nombres : étant donné un entier n qui est le produit de deux nombres premiers p et q, déterminer les facteurs p, q. Les records de factorisation sont rappelés dans le tableau suivant :

Digits	Date	Longueur (bits)
130	avril 1996	431 bits
140	février 1999	465 bits
155	août 1999	512 bits
160	avril 2003	531 bits
200	mai 2005	664 bits
232	décembre 2009	768 bits

Les deux résultats soulignés en rouge correspondent d'une part au célèbre record de 1999 concernant la factorisation d'entiers de la taille du module utilisé alors dans le

système RSA, et d'autre part au record le plus récent pour les entiers représentables sur 768 bits. L'estimation de la complexité, c'est-à-dire du coût calculatoire en terme du nombre d'opérations nécessaires en fonction de la taille des entiers, pour les algorithmes de factorisation donne :

768 bits $\rightarrow 2^{64}$ opérations	3072 bits $\rightarrow 2^{128}$ opérations
1024 bits $\rightarrow 2^{80}$ opérations	7680 bits $\rightarrow 2^{192}$ opérations
2048 bits $\rightarrow 2^{112}$ opérations	15360 bits $\rightarrow 2^{256}$ opérations

Le nombre d'opérations en rouge correspond à la puissance de calcul que l'on ne sait pas mettre en oeuvre actuellement. C'est donc cette *borne de complexité* que l'on choisit pour borner la puissance de calcul de l'adversaire. Bien sûr, cette borne évoluera avec le temps comme le montre le tableau précédent.

Dans les preuves de sécurité, les réductions peuvent avoir des coûts plus ou moins importants. On ne peut plus se contenter de parler de réduction en temps polynomiale comme en théorie de la complexité classique où la question est de montrer que deux problèmes sont polynomialement équivalents. Pour réussir à connaître de manière plus fine la dimension des paramètres à choisir en fonction du niveau de sécurité souhaité, il est nécessaire d'étudier concrètement le coût de la réduction. Si l'on suppose que l'attaque de l'adversaire fonctionne en temps t, le simulateur va interagir avec l'adversaire et fournir un algorithme global pour résoudre le problème difficile fonctionnant en un temps T, qui est une fonction $f(t)$ dépendant du temps t de l'attaque, suivant le nombre d'interactions avec l'adversaire et des calculs supplémentaires éventuels. Les réductions peuvent être soit *lâches* soit *fines* et les implications sur les tailles de paramètres seront plus ou moins importantes suivant la finesse de la réduction. On suppose, par exemple, que l'on a une réduction assez mauvaise qui, à partir d'un adversaire calculant en temps t, fournit un algorithme calculant en temps $T = f(t)$ avec une fonction $f(t) = k^3 \times t$ où k est le paramètre du schéma, comme le module du système RSA.

Longueur (bits) Module	Complexité Adversaire	Complexité Algorithme	Meilleure Complexité
$k = 1024$	$t < 2^{80}$	$T < 2^{110}$	2^{80}
$k = 2048$	$t < 2^{80}$	$T < 2^{113}$	2^{112}
$k = 3072$	$t < 2^{80}$	$T < 2^{115}$	2^{128}

On remarque que, pour un module de taille 1024, si le temps de l'attaque de l'adversaire est plus petit que la borne 2^{80} sur la puissance de calcul admise, le temps de l'algorithme global est seulement borné par 2^{110}, alors que le meilleur algorithme connu

de factorisation est de complexité plus petite. Il en est de même pour un module de taille 2048. C'est seulement en prenant un module de taille 3072 que l'on a une contradiction : la complexité de l'algorithme global serait bornée par 2^{115}, alors que celle du meilleur algorithme connu de factorisation est 2^{128}. Par contre, si l'on peut mettre au point une réduction *fine*, où le temps de calcul de l'algorithme global T est du même ordre que celui t de l'attaque de l'adversaire, il suffit d'un module de taille 1024 pour obtenir $T < 2^{80}$, en contradiction avec la complexité du meilleur algorithme connu de factorisation.

En conclusion, l'étude de la complexité algorithmique et, plus généralement des classes de complexité, peut s'avérer extrêmement utile :

- les mesures de complexité en temps de calcul et en espace permettent la comparaison des problèmes,
- de nombreux domaines d'application fournissent des exemples de problèmes intrinsèquement difficiles,
- la notion de réduction est fondamentale pour l'étude de la complexité et se trouve au coeur de la problématique $P = NP$,
- comme dans l'exemple des algorithmes de classement dans les moteurs de recherche, les algorithmes réellement efficaces font souvent appel aux méthodes d'approximation et aux algorithmes probabilistes,
- les notions de base de la complexité, problèmes difficiles, réduction et puissance de calcul, permettent de donner un cadre aux preuves de sécurité des protocoles cryptographiques.

Références

[1] Pavel Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1) :73–120, 2005.

[2] M.R. Garey et D.S. Johnson. *Computers and Intractability : a guide to NP-completeness*. Freeman and Co, 1979.

[3] Richard Lassaigne et Michel de Rougemont. *Logic and Complexity*. Springer Verlag, 2004.

[4] Michel Habib. *Les algorithmes de classement utilisés dans les moteurs de recherche*. Séminaire INRIA sur les objets numériques, 2009.

[5] Christos Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[6] David Pointcheval. *Quelles garanties avec la cryptographie ?* Collège de France, 2011.