Problème de Correspondance de Post

Chejjari Adnane Cordeiro Fonseca Loïc Leloup Loïc

Noubissi Kamgang Allan Perraudin Benjamin

Taheri Amirsaeed

Février 2022

Dans le cadre du cours "INFO-F308 - Projet transdisciplinaire" et de ce Printemps des Sciences, nous nous sommes penchés sur le Problème de Correspondance de Post, ou PCP, un problème introduit par Emile Louis Post en 1964.
Dans ce problème, on se donne un alphabet Σ de symboles fini, deux différentes listes A et B de taille k contenant des mots α_i et β_i composé des symboles des deux alphabets, telles que $i \in \{1, ..., k\}$. Le problème consiste en savoir si pour un ensemble donné de ces paramètres, il existe ou non une suite d'indices i, permettant de former à base des mots α_i et β_i deux mots identiques.
Il est important d'insister sur le fait que PCP ne s'intéresse pas à la suite d'indices même, mais à l'existence de celle-ci. En effet, PCP est un problème de décidabilité, on veut donc que savoir si la solution existe ou non. Ce qui rend PCP particulièrement intéressant est que ce problème est indécidable, c'est à dire qu'on ne peut pas pour tous les cas dire s'ils ont ont une solution ou non. En termes d'une machine, cela veut dire qu'il n'existe pas d'algorithme permettant de résoudre ce problème dans son entièreté.
Dans le cadre du Printemps des Sciences nous avons été tâché de présenter cette problématique relativement complexe et abstraite, à une audience de différents âges et sans connaissances à priori sur le sujet.

Pour cela nous avons développé un petit jeu permettant à une personne de voir différentes instances de PCP et de soi même tenter de voir si oui ou non ces instances ont une solution. Ainsi le joueur comprendra en premier lieu la décidabilité. Puis des cas plus difficiles mettront le joueur face à une impossibilité de répondre avec sûreté, les laissant indécis, ainsi ils seront confrontés directement à l'indécidabilité. Comme mentionné plus haut il n'existe pas d'algorithme nous permettant de résoudre le PCP, or afin de rendre le jeu correct, il nous fallait des instances de PCP dont on sait qu'elles sont solvables ou non. Il fallait donc trouver une façon de trouver des instances tout en sachant si ces instances sont solvable ou non. Pour cela nous avons développé un algorithme qui construit des blocs de symboles sur un alphabet donné, au hasard. On peut, en effet trouver des instances assez simple en procédant par un tel tâtonnement, mais c'est un processus assez lourd qui ne peut être fait en cours de jeu. Nous avons donc récolté un nombre d'instances trouvées avec cet algorithme pour peupler une base de données de laquelle le jeu tire les instances données.
Remarquez par contre que ce genre d'algorithme, bien qu'il nous donne la réponse qu'on cherche ne vérifie en soi pas la décidabilité du problème, mais cherche une solution, prouvant qu'une instance est solvable ou non. Cela permet de répondre pour des instance ayant une solution allant jusqu'à plus de 60 indices, ce qui résulte déjà en des mots de taille assez grande. Certaines instances auraient donc des solutions encore plus grandes, telles qu'on peut bien s'imaginer que certaines tendent vers l'infini, faisant en sorte qu'un tel algorithme ne termine jamais de chercher un solution et donc ne peut pas répondre au problème. Cet algorithme ne résout donc pas le PCP en son entièreté, mais reste le meilleur outil pour le résoudre.

C'est donc ainsi que nous vérifions les instances construites pour notre jeu, mais la construction des instances porte aussi quelques subtilités. En effet, afin d'avoir une instance pour laquelle on sait qu'elle a une solution ou non on pourrait penser à formuler un mot qui serait la solution et de scinder celui-ci de deux façon différentes avec le même nombre de parties. Cela nous donnerait donc une instance solvable, et nos deux listes A et B, et si on veut la rendre insolvable il suffirait de changer une lettre dans une des listes de sorte à ce qu'on ne puisse plus composer de mot.
Mais cette méthode porte un petit bémol, car elle fonctionne bien pour des simples cas, mais ne prend pas bien en compte les cas plus complexes où des mots des listes, pourraient être répétés plusieurs fois, or ce sont ces cas qui sont les plus intéressants et qui rendent une instance bien difficile. Nous créons donc des blocs au hasard et tentons de chercher une solution à partir de ces deux ensembles de blocs, par un backtracking assez brut. Afin de ne pas perdre de temps à explorer des instances qu'un humain ne saurait résoudre, nous avons borné la taille de solution à 20.