INFO-F—302 : Nombre d'Alcuin d'un graphe

Berthion Antoine - 566199

6 décembre 2024

1 Introduction

Ce rapport a pour objectif d'examiner les stratégies d'implémentation du problème d'Alcuin, formulé au IXe siècle, dans l'ouvrage *Ormesby Psalter*. À cette fin, nous utiliserons un **solveur SAT** ainsi que des outils permettant de formaliser ce problème sous forme d'une expression booléenne, précisément une CNF (conjonction de disjonctions).

2 Calcul du nombre d'Alcuin dans un graphe

Dans cette section, nous nous concentrerons sur les questions 1, 2 et 3 de l'énoncé du problème. Nous aborderons la formalisation en **CNF** (*Conjunctive Normal Form*) du problème général d'Alcuin, en prenant en compte :

— un nombre n de sujets et un nombre m de conflits entre ces sujets,

— la contrainte d'absence de conflits à bord du bateau.

Cette formalisation constituera une étape clé pour la résolution algorithmique du problème.

2.1 Formalisation en CNF

Afin de formaliser le problème en **CNF**, nous devons établir des contraintes sur nos littéraux, afin d'en faire une conjonction. Définissons en premier lieu les différentes fonctions de valuation que nous utiliserons.

— $x_{t,s,r} : K^3 \to \{0,1\}$, une fonction prenant en paramètres un moment t, un sujet s, et une rive $r \in \{0,1\}$ (où la rive gauche est notée 0 et la rive droite 1). La fonction renvoie la valeur 1 si le sujet s se trouve sur la rive r au temps t, et 0 sinon.

— $b_{t,r} : K^2 \to \{0,1\}$, un fonction prenant en paramètres un moment t, et une rive $r \in \{0,1\}$. La fonction renvoie la valeur 1 si le **berger** se trouve sur la rive r au temps t, et 0 sinon.

Grâce à ces fonctions de valuation, nous allons écrire une formule **CNF** générant une séquence valide (vérifiant la *conjonction* de **toutes** nos contraintes).

2.1.1 Contraintes initiale et terminale sur la séquence

En premier lieu, il est nécessaire d'exprimer que notre séquence commence par un état initial et se termine par un état terminal. Les contraintes associées à l'état initial peuvent être formalisées comme suit :

$$\phi = \bigwedge_{s \in \text{subjects}} (x_{t=0,s,r=0}) \wedge b_{t=0,r=0}.$$

Concernant l'état final, la taille de la séquence doit être prise en compte. Selon le théorème n°1 de l'énoncé [1], pour tout graphe G à n sommets, il existe une séquence correcte s de longueur au plus

1. Péter Csorba, Cor A. J. Hurkens and Gerhard J. Woeginger (2012). The Alcuin Number of a Graph and Its Connections to the Vertex Cover Number, SIAM. 6

$2n+1$ telle que Alcuin(G) = Alcuin(s). Nous considérons donc des séquences de $2n+2$ configurations. La contrainte finale s'exprime alors comme suit :

$$\phi = \bigwedge_{s \in \text{subjects}} (x_{t=2n+1,s,r=1}) \wedge b_{t=2n+1,r=1}.$$

2.1.2 Contrainte de transition

Nous souhaitons à présent créer une séquence possédant des transitions logiques. Intuitivement, il s'agit de dire que les sujets n'étant pas du côté du berger ne peuvent se déplacer. Plus formellement, on pose :

$$\phi = \bigwedge_{t=0}^{(2n+2)-1} \left(\bigwedge_{s \in \text{subjects}} ((\neg b_{t=t,r=0} \vee \neg x_{t=t,s,r=1} \vee x_{t=t+1,s,r=1}) \wedge (\neg b_{t=t,r=1} \vee \neg x_{t=t,s,r=0} \vee x_{t=t+1,s,r=0})) \right)$$

Cette formule en forme de **CNF** s'explique comme suit : pour chaque transition, nous imposons que si le berger se trouve sur la rive gauche, alors aucun sujet sur la rive droite ne peut se déplacer, et réciproquement. Les simplifications logiques découlant de l'implication simple permettent d'obtenir directement cette conjonction de disjonctions.

2.1.3 Contrainte d'unicité

Rapidement, vérifions l'unicité de chaque élément. En clair, nous voulons montrer qu'un sujet (ou le berger) ne peut se trouver que sur une et une seule rive. Il n'est donc pas possible d'avoir un sujet sur deux rives à la fois, ou sur aucune d'entre elles. On écrit alors la **CNF** suivante :

$$\phi = \bigwedge_{t=0}^{(2n+2)} \left(\bigwedge_{s \in \text{subjects}} ((x_{t,s,r=0} \vee x_{t,s,r=1}) \wedge (\neg x_{t,s,r=0} \vee \neg x_{t,s,r=1})) \wedge ((b_{t=t,r=0} \vee b_{t=t,r=1}) \wedge (\neg b_{t=t,r=0} \vee \neg b_{t=t,r=1})) \right)$$

2.1.4 Contrainte de déplacement du berger

Afin de maintenir une cohérence entre les configurations, nous souhaitons que le berger se déplacer de rive pour **chaque transition**. Cette contrainte s'exprime simplement de la manière qui suit :

$$\phi = \bigwedge_{t=0}^{(2n+2)-1} ((\neg b_{t=t,r=0} \vee b_{t=t+1,r=1}) \wedge (\neg b_{t=t,r=1} \vee b_{t=t+1,r=0}))$$

2.1.5 Contrainte de conflit

Afin d'aider à la formalisation du concept de conflit, discutons de la logique derrière la formule que nous allons écrire. Un conflit entre deux sujets, s_1 et s_2 existe si ces deux sommets sont liés par une arête du graphe. Pour chaque transition, nous voudrions donc vérifier que pour chaque paire de sujet (s_1, s_2) de la rive **ou le berger ne se trouve pas** ne représente pas une arête du graphe. On peut alors écrire formellement :

$$\phi = \bigwedge_{t=0}^{(2n+2)} \left(\bigwedge_{(s_1,s_2) \subseteq \text{subjects}} ((\neg b_{t=t,r=0} \vee \neg x_{t=t,s=s1,r=1} \vee \neg x_{t=t,s=s2,r=1}) \wedge (\neg b_{t=t,r=1} \vee \neg x_{t=t,s=s1,r=0} \vee \neg x_{t=t,s=s2,r=0})) \right)$$

Nos contraintes sont désormais formalisées. Nous souhaitons cependant poser une contrainte sur le nombre d'Alcuin des séquences respectant les contraintes posées. En effet, nous aimerions créer une contrainte supplémentaire posant que le nombre d'Alcuin d'une séquence devrait être inférieur à un entier positif k.

2.1.6 Contrainte d'Alcuin maximum

Comme discuté précédemment, nous souhaitons contraindre le nombre d'Alcuin des séquences valides, en le bornant par k. Comme il n'existe pas d'outil pour compter en logique booléenne, nous allons avoir besoin de recourir à un stratagème [2]. Intuitivement, si le nombre d'Alcuin est fixé entre 1 et k, nous pouvons dire que pour chaque transition, il y'a au plus k sujets de déplaçant.

Plus précisément, on peut affirmer que les sous-ensembles des sujets de déplaçant pour chaque transition sont de taille $\leq k$. On donne alors l'idée de créer chaque sous-ensemble de taille $k+1$, et de vérifier si les éléments de ces sous-ensembles se déplacent tous. Si tel est le cas, alors le nombre d'Alcuin dépasse k, ce qui violerait notre contrainte.

Cette astuce de comptage nous permet dès lors de vérifier que l'Alcuin ne dépasse pas un entier naturel k. On va définir la formule suivante :

$$\phi = \bigwedge_{t=0}^{(2n+2)-1} \bigwedge_{\substack{subg \subseteq subjects \\ |subg|=k+1}} \bigwedge_{s \in subg} \bigwedge_{r \in \{0,1\}} \left(\neg x_{t=t,s=s,r=r} \vee \neg x_{t=t+1,s=s,r=(r+1)\%2} \right)$$

De cette façon, seules les séquences d'Alcuin inférieur ou égal à k seront considérées comme valides.

2.2 Calcul du nombre d'Alcuin d'un problème donné

Après avoir formalisé notre formule en **CNF**, l'objectif est désormais de calculer le nombre d'Alcuin pour un problème donné.

De manière intuitive, le nombre d'Alcuin maximal pour un problème impliquant n sujets est n. Ce cas correspond à la situation où tous les sujets sont transportés simultanément sur le bateau. Par ailleurs, on peut raisonnablement supposer que le nombre d'Alcuin est au minimum égal à 1, car pour transporter des sujets d'une rive à l'autre, une place sur le bateau est nécessaire. Ainsi, en combinant ces intuitions, on en déduit que l'Alcuin d'un graphe quelconque G à n sommets doit être compris entre 1 et n.

Il semble donc raisonnable de vérifier, dans l'ordre croissant, l'existence d'une solution pour les valeurs de k allant de 1 à n. La première séquence trouvée pour un $k = i$ (voir la section 2) correspond alors à l'Alcuin de G, c'est-à-dire i.

3 Calcul du nombre d'Alcuin *c-valide*

Dans cette partie, nous allons discuter du cas spécifique du problème d'Alcuin, ou les conflits peuvent avoir occurence sur le bateau. Afin d'éviter cela, nous introduisons le concept des *partitions*, permettant de séparer les sujets en c groupes, afin d'éviter les conflits. On appelle une c-partition **stable** quand elle permet d'éviter les conflits sur le bateau.

3.1 Définition de la contrainte de c-validité

Afin de générer des solutions valides, nous devons ajouter une contrainte à celles définies dans la sous-section 2.1. En clair, nous devons poser une contrainte sur les transitions, en interdisant les transitions de sous-groupes n'étant pas **c-partitionnables**, c'est à dire ne possédant aucune **c-partition stable**.

Nous avons désormais besoin d'un algorithme générant toutes les c-partitions d'un ensemble de sujets. Nous aurons également besoin d'un second algorithme, permettant de déterminer si une c-partition est stable, pour une graphe G donné. Enfin, nous déterminerons pour chaque sous-ensemble de sujets sa c-partition stable. Si elle n'existe pas, alors il faudra contraindre pour chacune des transitions ce même

2. Notons qu'il existe des outils permettant de compter avec les outils SAT, mais nous ne nous en servirons pas ici, puisqu'il ne s'agit pas du but de l'exercice. Notre implémentation ne sera cependant pas optimisée de meilleure des manières, de ce fait.

sous-ensemble. Cependant, si une telle partition stable existe, alors elle ne violera aucune contrainte.

Notons p_sub l'ensemble des sous-ensembles de sujets n'ayant aucune c-partition valide. Nous devons contraindre les éléments de ces sous-ensembles, et ce pour chaque transition. La façon formelle, en forme **CNF**, ressemble en tout point à celle donnée dans la contrainte énoncée à la section 2.1.6. Elle s'énnonce comme suit :

$$\phi = \bigwedge_{s \subseteq p\_sub} \bigwedge_{t=0}^{(2n+2)-1} \bigwedge_{r \in \{0,1\}} \bigwedge_{i \in s} \left(\neg x_{t=t,s=i,r=r} \vee \neg x_{t=t+1,s=i,r=(r+1)\%2} \right)$$

3.2 Calcul du nombre d'Alcuin c-valide d'un problème donné

Maintenant que nous sommes en possession d'une formule booléenne permettant de déterminer la c-validité d'un Alcuin k, nous pouvons réutiliser la méthode de la section 2.2. Nous allons vérifier dans l'ordre croissant, pour k allant de 1 à n si une solution d'Alcuin k étant c-valide existe. Notons qu'évidemment, si $c = k$, alors de manière évidente le problème est c-valide pour un Alcuin $= k$.

4 Utilisation des LLM

Dans cette courte partie, nous discuterons de l'utilisation des LLM dans le cadre du projet. Aucun LLM n'a été utilisé pour l'aspect implémentation ainsi que la compréhension du projet. Cependant, ce rapport à été corrigé du point de vue de la syntaxe et de la grammaire par DeepL ainsi qu'un modèle GPT. Notons tout de même qu'aucune des informations du rapport n'a été produite par une autre personne que l'auteur.

5 Conclusion

En conclusion, nous dirons que nous avons décrit précisément et formellement la résolution du problème d'Alcuin, en utilisant un **solveur SAT** sur une formule en **CNF**. Ce projet nous a introduit aux solveurs SAT et nous a fait formaliser un problème à l'aide de la théorie des graphes et de la logique booléenne.