Bougies en réseaux

Pierre Duclosson, Lionel Richard

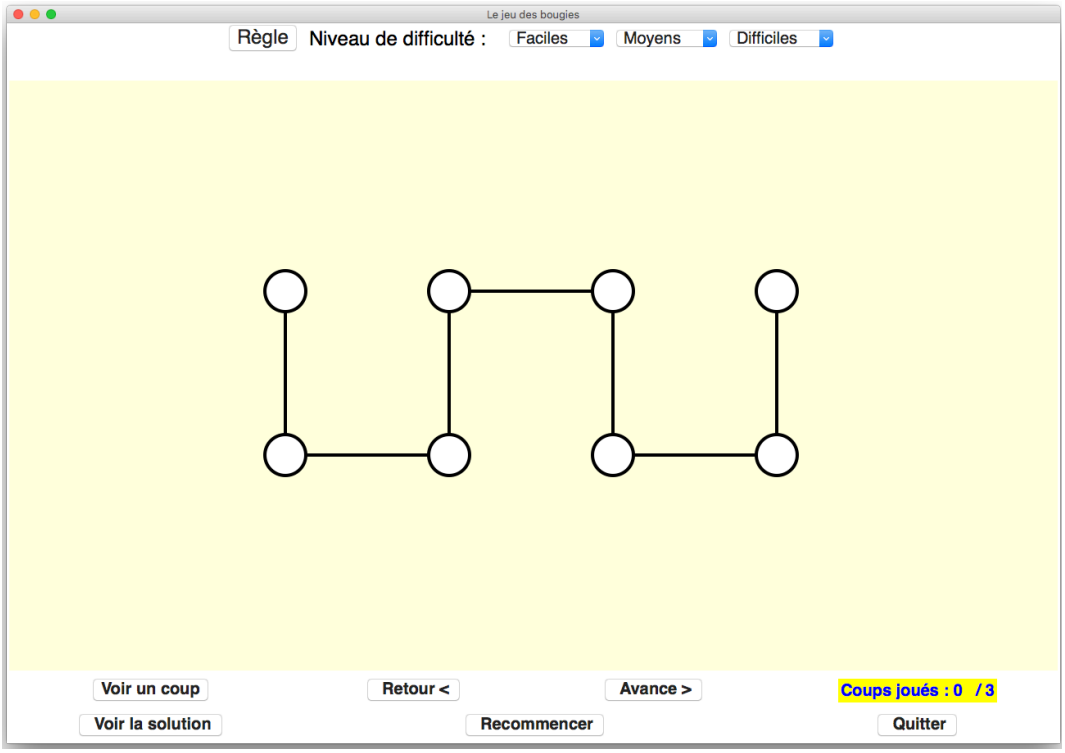
Formation ISN 2015-2016

I Présentation du projet, les interfaces graphiques

Le projet consiste en la réalisation d'une interface graphique permettant de jouer à un jeu de réflexion et d'une seconde interface qui sert à concevoir de nouveaux problèmes. Le jeu se joue sur un graphe plan dont les sommets peuvent présentés un état « alumé » ou « éteint ». Tous les sommets sont allumés au départ. En agissant sur un sommet on le fait passer d'un état à l'autre **ainsi que ses voisins immédiats**. Le but du jeu et de faire passer tous les sommets à l'état « éteint ». Chaque graphe différent est un nouveau problème.

I.1 L'interface de jeu (Pierre Duclosson)

La première partie du projet à permis la réalisation de l'interface de jeu suivante :



Dans sa version finale, elle permet de :

— sélectionner un problème grâce à une série de trois menus déroulants,
— obtenir l'affichage de la règle du jeu (bouton « Règle »),
— jouer en cliquant sur les sommets jusqu'à ce que le problème soit résolu [1],
— voir un coup qui rapproche de la solution [2],
— voir la solution [3],
— avancer et reculer dans les coups déjà joués [4],
— recommencer un problème,
— afficher le nombre de coups joués et le nombre minimal de coups possibles,
— quitter le jeu.

On peut se poser la question naturelle suivante : tout problème admet-il une solution ?

La réponse est oui. La démonstration de cette propriété fait appel à des éléments d'algèbre, elle est donnée en annexe.

Le nombre de solutions est une puissance de 2 comprise entre 1 et 2^{n-1} où n est le nombre de sommets.

I.2 L'interface de conception de nouveaux problèmes (Lionel Richard)

But : Concevoir un problème en plaçant dans une figure les éléments « sommets » et « arêtes ».

....
Partie élaguée car non réalisée par Pierre Duclosson
....

II Les trois parties du projet

II.1 Le développement de l'interface joueur (Pierre Duclosson)

Pour représenter un problème, j'ai choisi la donnée d'un tuple `(S, A)` avec :
— La liste `S` des coordonnées des sommets qui sont des couples, `S` est donc une liste de tuples.
— La liste `A` des arêtes qui sont chacune la donnée de deux numéros de sommets, `A` est aussi une liste de tuple.

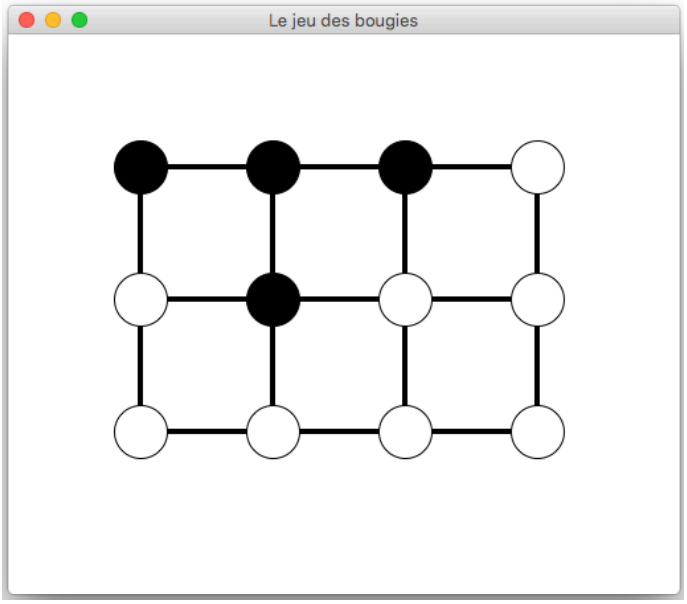
Par exemple : `(S, A)` avec `S = [(100, 100), (100, 300), (300, 300), (300, 100)]` et `A = [(0, 1), (1, 2), (2, 3), (3, 0)]` code le premier problème parmi les « faciles ».

Ce choix m'a permis de coder facilement les premiers problèmes à l'aide de papier et crayon. Il est cependant à noter qu'une fois l'affichage du graphe obtenue, la liste des arêtes est abandonnée au profit de la liste d'adjacence [5] qui s'en déduit.

Le développement s'est fait ensuite en suivant une **méthode incrémentale** en plusieurs phases répondant chacune à un **cahier des charges** plus ou moins précis :

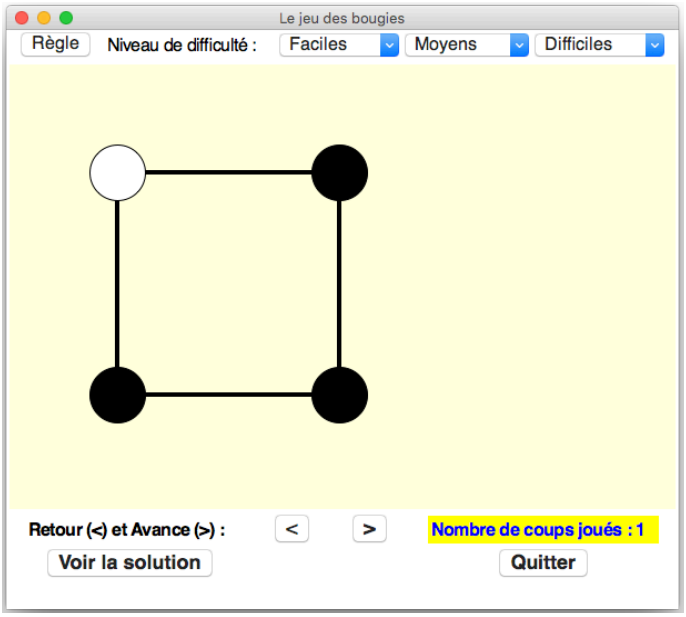
1. Ce qui provoque l'apparition d'un message de félicitation et empêche d'ajouter une nouvelle action.
2. Le bord du sommet passe en rouge.
3. Tous les sommet sur lesquels il faut agir sont signalés en rouge.
4. Ajouter un coup ré-initialise alors la mémoire des coups
5. Le calcule des solutions par l'algorithme du pivot de Gauss utilise aussi la matrice d'adjacence.

1. **Jouer sur un graphe**.



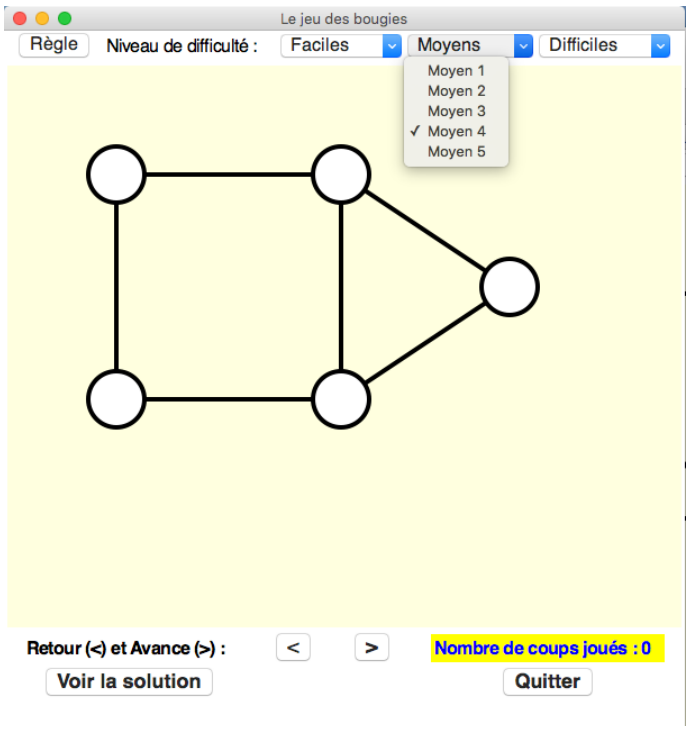
Il s'agissait d'obtenir l'affichage d'un unique graphe prédéterminé, modifier son apparence (suivant la règle des voisins) en cliquant sur les sommets. Cette phase m'a permis d'apprendre les rudiments d'utilisation des interfaces graphiques.

2. **Ajouter les widgets** correspondants aux différentes fonctionnalités envisagées.



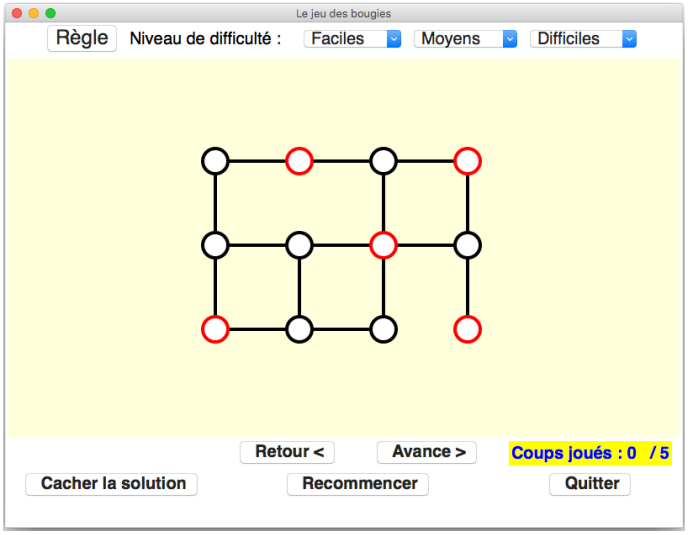
Je voulais que, quel que soit le système d'exploitation, le menu des problèmes soit proche du curseur de la souris du joueur. J'ai donc ajouté un cadre en haut de la fenêtre contenant ces éléments. À ce niveau les menus déroulants n'étaient pas encore actifs. Les éléments susceptibles d'être utilisés pendant le jeu sont regroupés sous l'espace où apparait le problème. Le nombre de coups est affiché au fur et à mesure.

3. Ajout des menus déroulants.



Au lancement du programme, le logiciel charge la liste des problèmes qui sont présent dans trois fichiers correspondant à chaque niveau et installe un élément de la liste de choix pour chaque problème. La sélection dans le menu correspondant fait apparaître le graphe du problème et on peut alors commencer à réfléchir au premier coup.

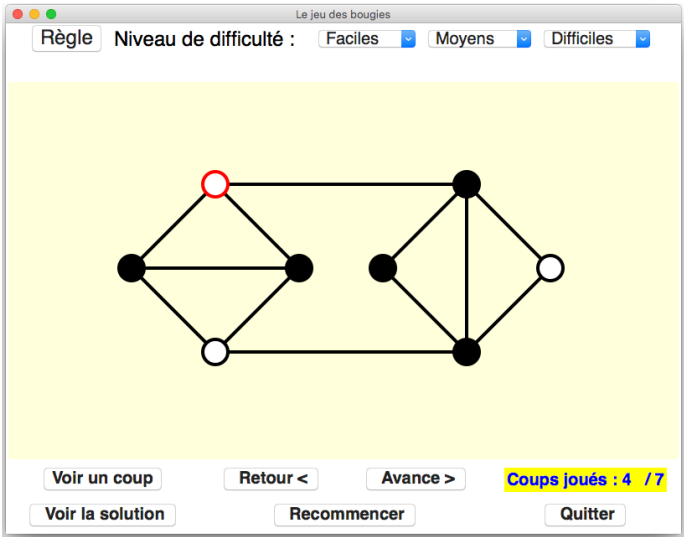
4. Ajout de la fonction « Voir la solution ».



J'ai implémenté un premier algorithme simple [6] qui détermine l'ensemble des solutions en testant toutes les possibilités d'action sur le graphe. Cela a permis de rendre actif le bouton permettant l'affichage de la solution. Le nombre minimal de coup est indiqué sous une barre de fraction au niveau de l'affichage du nombre de coups. Chaque action sur le graphe met à jour l'affichage de la solution.

6. De type « force brute ».

5. Deuxième algorithme, historique des coups et « Voir un coup ».



J'ai mis au point un second algorithme [7] basé sur celui du pivot de Gauss pour déterminer l'ensemble des solutions d'un problème en partant d'une situation initiale quelconque. J'ai aussi ajouté les éléments permettant d'activer les boutons « **Retour** < » et « **Avance** > » à l'aide d'une liste qui conserve l'historique des coups joués. Enfin le bouton « Voir un coup » permet d'obtenir une aide ponctuelle pour conserver son intérêt au jeu sans rester bloqué.

II.2 Le développement de l'interface de conception des nouveaux problèmes (Lionel Richard)

....
Partie élaguée car non réalisée par Pierre Duclosson
....

II.3 Deux algorithmes pour résoudre les problèmes (P. Duclosson)

Chaque algorithme donne l'ensemble des actions possibles sur l'ensemble du graphe pour passer d'un état initial à l'état final.

Détaillons le fonctionnement du **premier algorithme.**

Les variables utilisées sont :
— Une liste de listes `L` pour représenter la liste d'adjacence du graphe.
— Une liste d'entiers `E` pour représenter l'état des sommets du graphe.
— Une liste d'entiers `A` pour représenter les actions sur les sommets du graphe.

L'algorithme est codé à l'aide de quatre fonctions :

La fonction `agir(L, E, k)` permet de modifier l'état de E selon une action sur le sommet k.

```
def agir(L, E, k):

    for i in L[k]:
        E[i] = 1 - E[i]
```

7. Pour plus de détail voir plus loin.

La fonction `est_solution(L, A, etat_initial)` teste si, à partir de l'état initial, les actions données par `A` forment une solution.

```
def est_solution(L, A, etat_initial):

    n = len(L)
    E = list(etat_initial) # pour éviter l'effet de bord de la fonction agir

    for i in range(n):
        if A[i] == 1:
            agir(L, E, i)

    return sum(E) == n
```

L'ensemble des 2^n actions possibles sur le graphe correspond à toutes les écritures binaires des entiers de 0 à $2^n - 1$. Pour les générer, on utilise une fonction classique qui renvoie la liste des premiers digits binaires d'un entier.

```
def binaire(x, n):

    P = []

    for k in range(n):
        P.append(x%2)
        x = x//2

    return P
```

Il suffit alors de tester chaque possibilité et d'ajouter celles qui sont valides à une liste. La fonction `solutions(L, etat_initial):` s'en charge. Elle renvoie une liste de listes de toutes les actions menant à la solution.

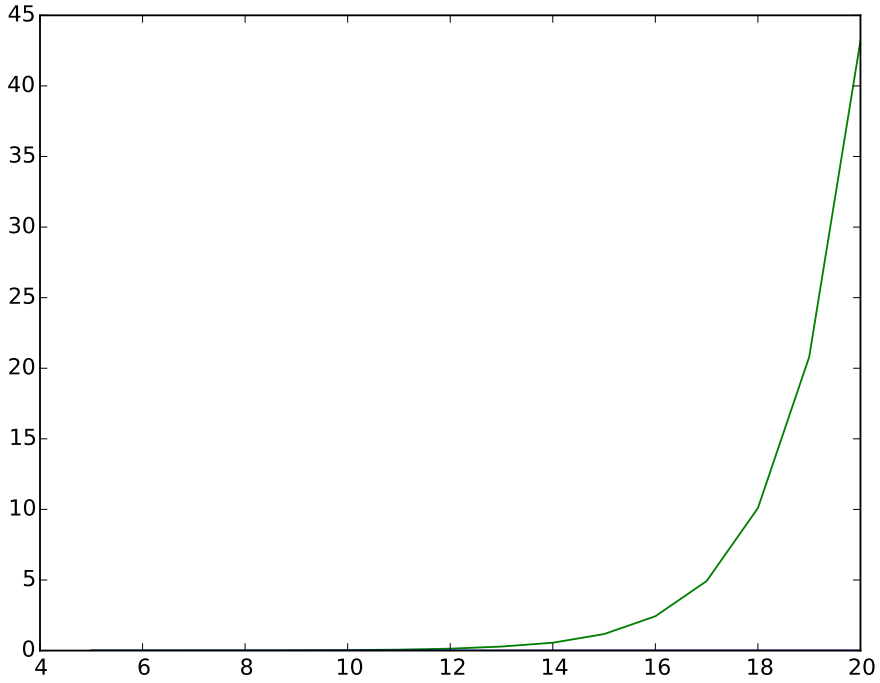
```
def solutions(L, etat_initial):

    n = len(L)
    S = []

    for i in range(2**n):  # Boucle qui va générer toutes les actions possibles
                           # sur l'ensemble du graphe
        A = binaire(i, n)  # A est l'action correspondant à l'entier i
        if est_solution(L, A, etat_initial):
            S.append(A)

    return S
```

Le premier algorithme n'est pas satisfaisant du point de vue de la complexité : sous l'hypothèse naturelle que le nombre de voisins de tout sommet est borné et en notant n le nombre de sommets d'un graphe, l'algorithme par la force brute est d'une complexité en $O(n2^n)$. Pour

un nombre relativement important de sommets [8] cela risque de poser un problème. Pour une interface graphique de ce type un ralentissement même modéré n'est pas acceptable si on veut conserver une expérience agréable pour l'utilisateur.

L'étude mathématique montre que la résolution d'un problème est équivalente à celle d'un système d'équation dans le corps $\mathbb{Z}/2\mathbb{Z}$. Cela permet de programmer un deuxième algorithme. C'est l'algorithme classique du pivot de Gauss à ceci prêt qu'il faut générer *explicitement* toutes les solutions, ce qui est possible car le corps de base est fini de cardinal 2. En supposant que la dimension de l'espace des solutions est bornée, l'algorithme est de même complexité que celui du pivot de Gauss, c'est à dire en $O(n^3)$. C'est nettement mieux que la complexité de l'algorithme par la force brute.

Le résultat de quelques simulations montrent l'écart entre les deux algorithmes :

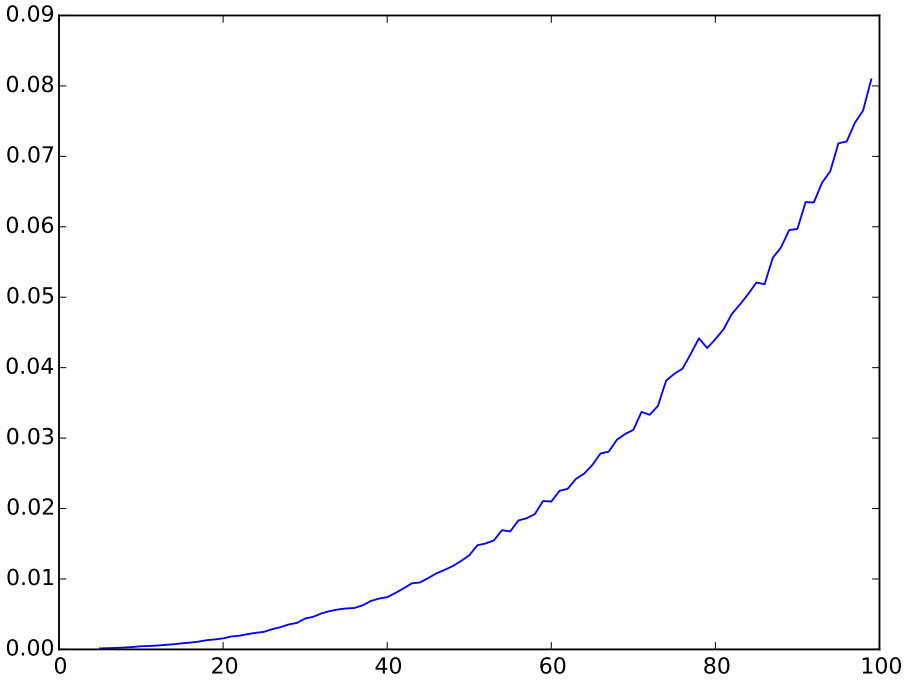


La courbe verte représente le temps d'exécution [9] en secondes de l'algorithme par la force brute pour des graphes ayant entre 5 et 20 sommets. Les temps d'exécution de l'algorithme par le pivot de Gauss sont affichés mais ils sont confondus avec l'axe des abscisses sur cette figure.

Sur cette seconde figure, seule les temps d'exécution de l'algorithme par le pivot de Gauss sont affichés :

8. L'expérience montrera qu'à partir de 15 sommets il y a un ralentissement sensible de la réaction de l'interface en mode « Voir la solution ».

9. Moyenne des temps pour 10 graphes choisis aléatoirement.



III Bilan pédagogique de l'expérience

Le travail réalisé m'a donné une certaine expérience. Je pense qu'elle me permettra d'anticiper sur l'encadrement des projets d'élèves. Pour cela il semble nécessaire de donner l'habitude de **bonnes pratiques pour le codage** en insistant dès le début d'année sur :

1. la nécessité d'écrire un code lisible en respectant l'esprit du standard PEP8,
2. séparer les partie d'initialisation et de traitement (boucles) par une ligne vide,
3. nommer de manière pertinente et homogène les variables et les fonctions,
4. ajouter des docstrings à chaque fois que nécessaire.

Il est nécessaire d'habituer les élèves à une **conception modulaire des programmes** car celle-ci n'est pas innée. Les devoirs à la maisons gagneront à être conçus en respectant cette démarche.

Il faut aussi saisir toutes les occasions de mener une réflexion sur la pertinence de telle ou telle représentation des données.

On peut développer progressivement en 3 ou 4 étapes et avec l'ensemble de la classe un mini-projet dont le sujet serait imposé pour **illustrer par l'exemple la méthodologie** à mettre en oeuvre lors des projets comptant pour le bac.

Remerciements

Je tiens à remercier Jean-Manue Meny pour ses remarques pertinentes qui m'ont permis d'améliorer plusieurs éléments de l'interface du jeu.

IV Annexe : Tout problème admet une solution

Dans ce qui suit n désigne le nombre de sommets.

On utilise la structure de l'anneau $\mathbb{Z}/2\mathbb{Z}$. Pour chaque sommet, on représente par 0 son état « alumé » et par 1 son état « éteint ». Un changement de son état correspond à l'addition de l'unité :

Partant de « alumé » un changement d'état donne « éteint » : $0 + 1 = 1$
Partant de « éteint » un changement d'état donne « alumé » : $1 + 1 = 0$

Une absence de changement correspond à l'ajout de zéro ainsi une suite de deux changements laisse l'état inchangé : $1 + 1 = 0$.

On représente l'état de l'ensemble des sommets par un vecteur de $(\mathbb{Z}/2\mathbb{Z})^n$. Par exemple, pour $n = 4$, si les sommets 2 et 3 sont allumés et les sommets 1 et 4 éteints, le vecteur $E = (1, 0, 0, 1)$ représente l'état du graphe.

Lorsque l'on agit successivement sur plusieurs sommets, pour chaque sommet seule la parité du nombre d'actions compte. Un ensemble d'action sur différents sommets est représenté par un autre vecteur. Par exemple, agir sur le sommet 1 et 2 mais pas sur 3 ni sur 4 correspond à $A = (1, 1, 0, 0)$. Le vecteur C donne les changements d'état pour l'ensemble des sommets qui résultent de ces actions. Il est obtenu en faisant le produit $C = MA$ où M est la matrice d'adjacence du graphe.

Pour résumer : si E_i représente l'état initial du graphe, que l'on agit sur les sommets donnés par le vecteur A alors l'état final du graphe est donné par :

$$E_f = E_i + MA$$

Pour montrer qu'il existe une solution, il suffit de prouver le lemme suivant :

Lemme :

Pour M une matrice symétrique de taille n, à coefficient dans $\mathbb{Z}/2\mathbb{Z}$ et qui porte des 1 sur sa diagonale, le vecteur $U = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ est dans $\mathrm{Im}(M)$.

Preuve :

On note $M = (m_{i,j})_{\substack{1 \leqslant i \leqslant n \\ 1 \leqslant j \leqslant n}}$. Pour tout vecteur $X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ de $(\mathbb{Z}/2\mathbb{Z})^n$, la caractéristique 2 et l'identité $x^2 = x$ qui a lieu dans $\mathbb{Z}/2\mathbb{Z}$ donne :

$$ {}^tXMX = \sum_{\substack{1 \leqslant i \leqslant n \\ 1 \leqslant j \leqslant n}} x_i m_{i,j} x_j = \sum_{i=1}^{n} x_i m_{i,i} x_i + 2 \sum_{1 \leqslant i < j \leqslant n} x_i m_{i,j} x_j = \sum_{i=1}^{n} {x_i}^2 = \sum_{i=1}^{n} x_i = {}^tXU $$

$\mathrm{Im}(M)$ est un sous-espace vectoriel de $(\mathbb{Z}/2\mathbb{Z})^n$, il peut se caractériser par un système d'équations. Il existe donc des vecteurs $X_1, \dots X_q$ tels que :

$$ X \in \mathrm{Im}(M) \iff \begin{cases} {}^tX_1X &= 0 \\ &\vdots \\ {}^tX_qX &= 0 \end{cases} $$

Le vecteur U vérifie chacune de ces équations car ${}^tX_iU = {}^tX_iMX_i = 0$ car MX_i est dans $\mathrm{Im}(M)$. On en déduit que U est dans $\mathrm{Im}(M)$.