D.S. 03

Couplages, mines 2012

1 Généralités

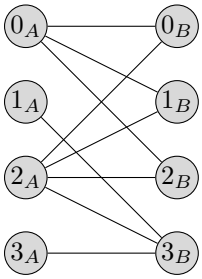


Figure I.1 – Le graphe G_0.

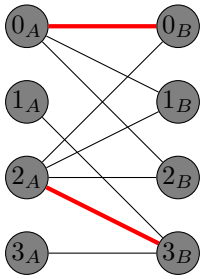


Figure I.2 – Le graphe G_0 et le couplage C_0.

Question 1
Exhiber un couplage de cardinal 3 de G_0, puis indiquer s'il existe dans G_0 un couplage de cardinal 4. Justifier la réponse.

Solution de la question 1 $\big\{\{0_A, 0_B\}, \{1_A, 3_B\}, \{2_A, 1_B\}\big\}$ est un couplage de cardinal 3 de G_0. Il n'existe pas de couplage de cardinal 4 car les sommets 1_A et 3_A ne peuvent être couplés qu'avec 3_B, ainsi un seul de ces deux sommet peut appartenir à couplage, tout couplage ne peut concerner que 3 sommets de A.

Question 2
Écrire une fonction `verifie` *telle que si* `g` *est une matrice codant le graphe G et* `c` *un tableau codant le tableau C alors* `verifie g c` *renvoie* `true` *si le tableau C représente un couplage dans G et* `false` *sinon. Indiquer la complexité de la fonction* `verifie`*.*

Solution de la question 2 Pour chaque nouveau couplage rencontré il faut vérifier si l'arête correspondante existe dans le graphe et si elle n'est pas incidente à une arête déjà rencontrée. Pour cela on utilise un tableau `vu` qui indique si un sommet de B a déjà été rencontré.

```
let verifie g c =
  let n = Array.length c in
  let reponse = ref true in
  let vu = Array.make n false in
  for i = 0 to (n-1) do
    let j = c.(i) in
    if j <> -1 && not vu.(j)
    then begin
           reponse := !reponse && g.(i).(j);
           vu.(j) <- true end done;
  !reponse;;
```

La complexité est linéaire en n, le nombre de sommets.

Question 3
Écrire une fonction `cardinal` *telle que si* `c` *est un tableau codant un couplage alors* `cardinal c` *renvoie le cardinal de ce couplage. Indiquer la complexité de la fonction* `cardinal`*.*

Solution de la question 3

```
let cardinal c =
  let n = Array.length c in
  let reponse = ref 0 in
  for i = 0 to (n-1) do
    if c.(i) <> -1
    then reponse := !reponse +1 done;
  !reponse;;
```

La complexité est linéaire en n.

2 Un couplage maximal

Question 4
Appliquer `algo_approche` *au graphe* G_0*.*

Solution de la question 4 Les degrés dans A sont $3, 1, 4, 1$; les degrés dans B sont $2, 2, 2, 3$.

1. Les arêtes de somme minimum 4 sont $(1, 3)$ et $(3, 3)$. On choisit par exemple, $(1, 3)$.
2. Les degrés dans A et B sont alors $3, \_, 3, 0$ et $2, 2, 2, \_$.
3. Toutes les arêtes restantes ont pour somme 5. On choisit par exemple, $(0, 0)$.
4. Il ne reste que $(2, 1)$ et $(2, 2)$ de somme 3 chacune. On choisit par exemple, $(2, 1)$.
5. Le graphe restant n'a plus d'arêtes.
6. On obtient ainsi $C = \{(1, 3), (0, 0), (2, 1)\}$ qui est bien maximal.

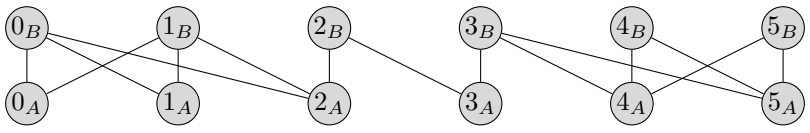
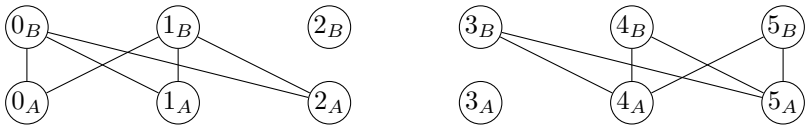


FIGURE I.3 – Le graphe G_1.

Question 5
Déterminer la première arête a_1 *choisie par* `algo_approche` *appliqué à* G_1 *;*
tracer le graphe obtenu après suppression de a_1 *et des arêtes incidentes à* a_1*.*
Montrer que le couplage obtenu par `algo_approche` *est de cardinal au plus* 5
et indiquer s'il est de cardinal maximal parmi les couplages de G_1*.*

Solution de la question 5 Dans G_1 toutes les arêtes ont une somme des degrés des extrémités égale à 4, 5 ou 6. Une seule a une somme égale à 4 : l'arête $a_1 = \{3_A, 2_B\}$. Une fois celle-ci éliminée ainsi que les arêtes incidentes il reste le graphe :



Chacune de deux composantes connexes du graphe contient 2 couplages au maximum donc l'algorithme retournera un couplage de cardinal inférieur ou égal à 5. Or il existe un couplage de cardinal 6 : $\{\{0_A, 0_B\}, \{1_A, 1_B\}, \{2_A, 2_B\}, \{3_A, 3_B\}, \{4_A, 4_B\}, \{5_A, 5_B\}\}$; l'algorithme ne garantit donc pas d'obtenir un couplage de cardinal maximal.

Question 6
Écrire une fonction `arete_min` *qui détermine une arête de G dont la somme des degrés des extrémités soit minimum. Si le graphe possède au moins une arête, cette fonction modifie le tableau a de deux entiers reçu en paramètre pour mettre dans les deux cases de a les numéros des deux extrémités d'une arête qui atteint ce minimum ; dans ce cas la fonction renvoie la valeur* `true` *; sinon elle renvoie la valeur* `false`*. Indiquer la complexité de la fonction* `arete_min`*.*

Solution de la question 6 On commence par rédiger une fonction qui calcule les degrés des sommets de A et des sommets de B :

```
let calcule_degre g =
  let n = Array.length g in
  let degA = Array.make n 0 in
  let degB = Array.make n 0 in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j)
      then begin degA.(i) <- degA.(i) + 1;
                 degB.(j) <- degB.(j) + 1 end done done;
  degA, degB;;
```

Cette première fonction est de complexité quadratique.
Il reste ensuite à partir à la recherche de l'arête minimale :

```
let arete_min g a =
  let n = Array.length g in
  let degA, degB = calcule_degre g in
  let s = ref (2 * n + 1) in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      if g.(i).(j) && degA.(i) + degB.(j) < !s
      then begin s := degA.(i) + degB.(j);
                 a.(0) <- i;
                 a.(1) <- j end
    done
  done;
  !s <= 2 * n;;
```

On notera que la somme des degrés des extrémités d'une arête ne peut excéder $2n$, ce qui explique la valeur initiale de la référence `s`.
Cette fonction est aussi de complexité quadratique.

Question 7
Écrire une fonction `supprimer` *telle que si* `g` *est une matrice codant un graphe biparti équilibré* G *et*`a` *un tableau de deux entiers codant une arête* a *de* G *alors* `supprimer g a` *modifie* `g` *pour que, après modifications,* `g` *code le graphe obtenu à partir de* G *en supprimant* a *ainsi que toutes les arêtes incidentes à* a.

Solution de la question 7

```
let supprimer g a =
  let n = Array.length g in
  let i = a.(0) and j = a.(1) in
  for k = 0 to n-1 do
    g.(i).(k) <- false;
    g.(k).(j) <- false done;;
```

Cette fonction est de coût linéaire.

Question 8
Écrire une fonction `algo_approche` *telle que si* `g` *est une matrice qui code un graphe biparti équilibré* G, `algo_approche` *effectue* `algo_approche` *à partir d'une copie de* G *et renvoie un tableau codant le couplage obtenu. Indiquer la complexité de la fonction* `algo_approche`.

Solution de la question 8 La fonction `Array.copy` réalise la copie d'un tableau, on peut d donc écrire

```
let copy_matrix m =
  let n = Array.length m in
  Array.init n (fun i -> Array.copy m.(i));;
```

On définit la fonction principale de l'algorithme *algo_approche* :

```
let algo_approche g =
  let gg = copy_matrix g in
  let n = Array.length g in
  let c = Array.make n (-1) in
  let a = [| 0; 0 |] in
  while arete_min gg a do
    c.(a.(0)) <- a.(1);
    supprimer gg a done;
  c;;
```

Sachant que le coût de la fonction `arete_min` est un $\mathcal{O}(n^2)$ et qu'un couplage maximal comporte au maximum n couplages, le coût total de cet algorithme est en un $\mathcal{O}(n^3)$

3 Recherche exhaustive

Question 9
Écrire une fonction `une_arete` *telle que si* `g` *est une matrice codant un graphe G et* `a` *un tableau de deux entiers alors* `une_arete g a` *modifie le tableau* `a` *en y inscrivant les indice d'une arête dans le cas où G possède au moins une arête.*

Solution de la question 9

```
let une_arete g a =
  let n = Array.length g in
  let trouve = ref false in
  let i = ref 0 in
  let j = ref 0 in
  while !i < 9 && not !trouve do
    if g.(!i).(!j)
    then begin a.(0) <- !i;
               a.(1) <- !j;
               trouve := true end
    else begin if j = 8 then (i := !i +1; j := 0)
                        else (j := !j + 1) end done;
  !trouve;;
```

Question 10
Écrire une fonction récursive `meilleur_couplage` *telle que, si* `g` *est une matrice codant un graphe biparti équilibré G,* `meilleur_couplage g` *renvoie un tableau codant un couplage de cardinal maximal dans G.*

Solution de la question 10 Lorsque le graphe G contient au moins une arête a, on réalise deux copies G_1 et G_2 de G. Dans la première on supprime l'arête a et dans la seconde l'arête a ainsi que toutes les arêtes incidentes.
Tout couplage C_1 de G_1 est un couplage de G ne contenant pas l'arête a ; tout couplage C_2 de G_2 à qui on ajoute a est un couplage de G contenant l'arête a. Ceci conduit à l'algorithme suivant :

```
let rec meilleur_couplage g =
  let n = Array.length g in
  let a = [| 0; 0 |] in
  if une_arete g a
  then begin
      let g1 = copy_matrix g in
      g1.(a.(0)).(a.(1)) <- false;
      let g2 = copy_matrix g in
      supprimer g2 a;
      let c1 = meilleur_couplage g1 in
      let c2 = meilleur_couplage g2 in
      c2.(a.(0)) <- a.(1);
      if cardinal c1 < cardinal c2 then c2 else c1
    end
  else Array.make n (-1);;
```

Les complexités temporelles et spatiales sont monstrueuses : de l'ordre de $n^2.2^{n^2}$.

4 L'algorithme hongrois

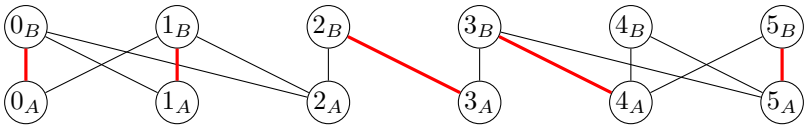


FIGURE I.4 – Le graphe G_1 et le couplage C_1

Question 11
Après avoir indiqué le seul sommet de A qui puisse être l'origine d'une chaîne alternée augmentante relativement à C_1 et le seul sommet B qui puisse être l'extrémité d'une chaîne alternée augmentante relativement à C_1, déterminer une chaîne alternée augmentante relativement à C_1.

Solution de la question 11 Le seul sommet non couplé de A est le sommet 2_A ; il doit être au départ de toute chaîne alternée relativement à C_1. Le seul sommet non couplé de B est le sommet 4_B ; il doit être à l'arrivée de toute chaîne alternée augmentante relativement à C_1.
Une chaîne alternée augmentante peut être $2_A, 2_B, 3_A, 3_B, 4_A, 4_B$.

Question 12
On considère un graphe biparti équilibré G et un couplage C dans G. Montrer que s'il existe une chaîne alternée augmentante relativement à C alors il existe dans G un couplage dont le cardinal est égal au cardinal de C augmenté de 1.

Solution de la question 12 Supposons que $(x_0, x_1, \ldots, x_{2p+1})$ soit une chaîne alternée augmentante relativement à un couplage C. Soit le couplage C' obtenu à partir de C en supprimant les p arêtes (x_{2k+1}, x_{2k+2}) pour $0 \leqslant k < p$ et en ajoutant les $p+1$ arêtes (x_{2k}, x_{2k+1}) pour $0 \leqslant k \leqslant p$.
Les sommets $x_1, x_2, \ldots, x_{2p}$ ont perdu leur couplage en ôtant les arêtes de la forme (x_{2k+1}, x_{2k+2}) donc on peut les coupler de nouveau en ajoutant aussi les sommets x_0 et x_{2p+1}.
Les arêtes ajoutées sont bien dans le graphe par définition d'une chaîne alternée donc on obtient bien un couplage de graphe. De plus son cardinal a été augmenté de 1.

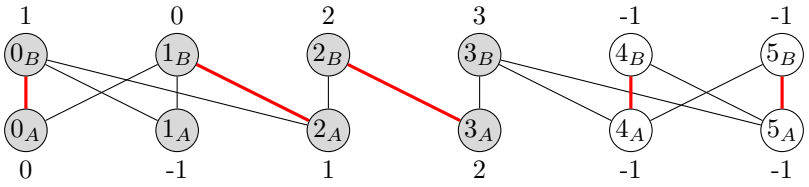


FIGURE I.5 – Le graphe G_1 et le couplage C_1', les sommets atteints sont grisés

Question 13
Utiliser les marques pour reconstituer la chaîne alternée arrivant dans le sommet 3_B et correspondant aux marques. Indiquer s'il s'agit d'une chaîne alternée augmentante relativement à C_1'.

Solution de la question 13 En partant de l'extrémité $3B$, on remonte la chaîne grâce aux marques. On obtient ainsi la chaîne alternée $(1A, 0B, 0A, 1B, 2A, 2B, 3A, 3B)$. Comme $3B$ n'est pas couplé dans C_1 et appartient à B, il s'agit bien d'une chaîne alternée augmentante de C_1.
On augmente alors le couplage en $(0A, 1B), (1A, 0B), (2A, 2B), (3A, 3B), (4A, 4B), (5A, 5B)$.

Question 14
Écrire une fonction `actualiser` *telle que si* `c`, `r`, `mA`, `mB` *sont des tableaux de n entiers qui correspondent à un couplage, sa réciproque et les tableaux des marques d'une chaîne alternée augmentante et si* `numero` *est un entier donnant le numéro de x_p alors* `actualiser c r mA mB numero` *modifie les tableaux* `c` *et* `r` *pour obtenir un couplage de cardinal égal à celui de C augmenté de 1.*

Solution de la question 14 On remonte la chaîne augmentante grâce aux marques en partant de l'extrémité x_p sous la forme d'un indice j tel que $x_p = j$.

- On lit `i = mB[j]` ; on crée l'arête (i, j) dans le couplage.
- Pour cela on place i dans `R[j]` et j dans `C[i]`
- On lit une nouvelle valeur de j dans `mA[i]`
- On recommence si j est différent de -1.

```
let rec actualiser cpl inv mA mB numero =
  let i = mB.(numero) in
  cpl.(i) <- numero;
  inv.(numero) <- i;
  let j = mA.(i) in
  if j <> -1
  then actualiser cpl inv mA mB j;;
```

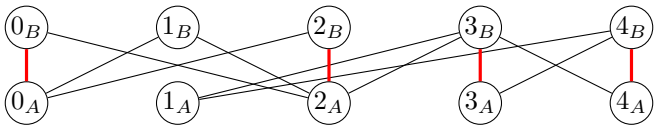


FIGURE I.6 – Le graphe G_2 et le couplage C_2

Question 15
Recopier la figure, encadrer tous les sommets qui peuvent être atteints et préciser à côté des sommets les marques obtenues.
Indiquer s'il existe dans G_2 une chaîne alternée augmentante relativement à C_2.

Solution de la question 15

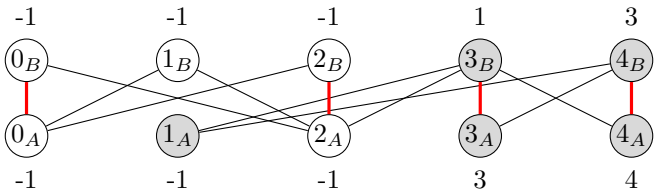


FIGURE I.7 – Le graphe G_2 et le premier marquage

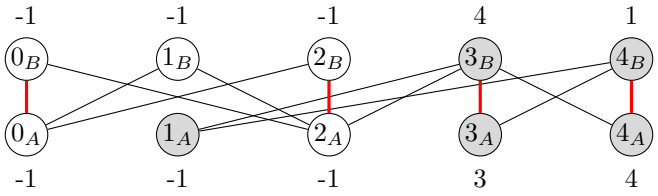


FIGURE I.8 – Le graphe G_2 et le second marquage

Les deux seules chaînes alternées (maximales) sont $(1A, 3B, 3A, 4B, 4A)$ et $(1A, 4B, 4A, 3B, 3A)$; aucune n'est augmentante.

Question 16
Définir ce qu'on appelle récursivité croisée et indiquer comment elle peut être implémentée en `OCaml`*, puis écrire les deux fonctions* `chercherA` *et* `chercherB` *; chacune de ces deux fonctions reçoit en paramètres une matrice* `g` *codant le graphe G, les quatre tableaux* `c`, `r`, `mA`, `mB` *et un entier codant le numéro du sommet de départ de la recherche.*
Ces deux fonctions modifient les tableaux `mA` *et* `mB` *et renvoient le numéro d'un sommet non couplé de B ou la valeur -1 selon le cas.*

Solution de la question 16 La récursivité croisée consiste à déterminer en même temps deux fonctions dont chacune fait appel à l'autre. Elles sont définies par un même `let` et leurs définitions sont séparées par un `and`.

On commence par chercher la liste des voisins d'un sommet de A :

```
let voisinsB i g =
  let n = Array.length g in
  let l = ref [] in
  for j = 0 to (n-1) do
    if g.(i).(j) then l := j::(!l) done;
  !l;;
```

La recherche à partir de B est simple :

- si le sommet est couplé, on suit le couplage
- sinon on a trouvé un point

Pour la recherche à partir de A

- (2) : on définit une fonction auxiliaire qui parcourt une liste de sommets adjacents à i
- (3) : s'il n'y en a plus on renvoie -1
- (5) : si un sommet n'est pas couplé à i et n'est pas marqué, on teste en (7)
  sinon on continue la recherche en (13)
- (7-8) : on le marque avec i et on cherche (dans B) à partir de lui
- (9-10) : si cela n'aboutit pas on dé-marque le sommet et on continue la recherche
- (11) : si la recherche aboutit on a trouvé

```
1  let rec chercherA g c inv mA mB i =
2    let rec aux liste =
3      match liste with
4      |[] -> -1
5      |t::q -> if t <> c.(i) && mB.(t) = -1
6               then begin mB.(t) <- i;
7                    let rep = chercherB g c inv mA mB t in
8                    if rep = -1
9                    then (mB.(t) <- -1; aux q)
10                   else rep end
11              else aux q in
12   aux (voisinsB i g)
13 and chercherB g c inv mA mB j =
14    match inv.(j) with
15    |(-1) -> j
16    |i -> mA.(i) <- j;
17          chercherA g c inv mA mB i;;
```

Question 17
Écrire une fonction `chaine_alternee` *telle que si* `g` *est une matrice codant le graphe* G, `c`, `r`, `mA` *et* `mB` *des tableaux, toutes les cases de* `mA` *et* `mB` *étant initialisées à* -1, *alors* `chaine_alternee g c r mA mB` *renvoie :*

- -1 *s'il n'existe pas de chaîne alternée augmentante ;*
- *le numéro de l'extrémité d'une chaîne alternée augmentante dans le cas contraire.*

De plus, la fonction modifie les tableaux `mA` *et* `mB` *pour qu'ils contiennent les marques des sommets à la fin de l'exécution de la fonction.*

Solution de la question 17 On parcourt dans l'ordre les sommets de A jusqu'à trouver un éventuel sommet non couplé origine d'une chaîne alternée augmentante. Ne pas oublier de réinitialiser les tableaux de marquage à chaque étape.

```
let chaine_alternee g c inv mA mB =
  let n = Array.length c in
  let rec aux i =
    if i = n
    then -1
    else begin for k = 0 to (n-1) do mA.(k) <- -1 done;
        for k = 0 to (n-1) do mB.(k) <- -1 done;
        let j = chercherA g c inv mA mB i in
        if j = -1 then aux (i+1) else j end in
  aux 0;;
```

Question 18
Écrire la fonction `algorithme_hongrois` *telle que si* `g` *est une matrice codant un graphe biparti équilibré* G, *alors* `algorithme_hongrois g` *renvoie un tableau codant le couplage obtenu par l'algorithme hongrois.*

Solution de la question 18 Le programme consiste

- à partir d'un couplage (le premier étant le couplage vide)
- on marque les sommets (`mA` et `mB`) avec -1
- on cherche une chaîne alternée augmentante avec `chaine_alternee`, cela modifie `mA` et `mB`
- s'il en existe une on augmente de couplage avec `actualiser`, cela modifie `c` et `inv` et on recommence

L'algorithme termine car a chaque tour de boucle le cardinal du couplage augmente de une unité : la boucle effectue au plus n passages.

```
let algorithme_hongrois g =
  let n = Array.length g in
  let c = Array.make n (-1) in
  let inv = Array.make n (-1) in
  let encore = ref true in
  while !encore do
    let mA = Array.make n (-1)
    and mB = Array.make n (-1) in
    let num = chaine_alternee g c inv mA mB in
    if (num <> -1)
    then actualiser c inv mA mB num
    else encore := false done;
  c;;
```