Option informatique — Mines-Ponts 2008

Partie I — Deux constructions d'un automate reconnaissant l'intersection

Question I.1

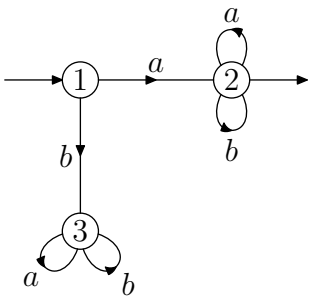

Figure 1 l'automate A_{1_ex}

Question I.2

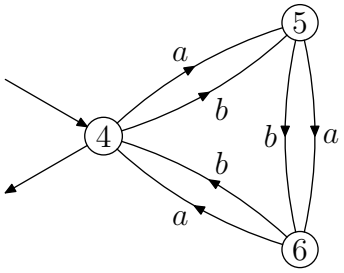

Figure 2 l'automate A_{2_ex}

Question I.3
Il s'agit de l'automate produit ; son ensemble T de transitions est constitué de la façon suivante : pour toutes transitions $(p_1, x, q_1) \in T_1$ et $(p_2, x, q_2) \in T_2$ étiquetées par la même lettre x, on crée une transition $(p_1, p_2) \xrightarrow{x} (q_1, q_2)$ dans T.
Un calcul réussi de l'automate A se projette sur la première composante en un calcul réussi de A_1 et sur la seconde en un calcul réussi de A_2, puisqu'on a choisi $F = F_1 \times F_2$: c'est-dire que les mots reconnus par A sont dans l'intersection $L_1 \cap L_2$.
Réciproquement, soit $x = x_1 x_2 \dots x_k$ un mot de $L_1 \cap L_2$. Il existe un calcul réussi de A_1 qui s'écrit $p_0 \xrightarrow{x_1} p_1 \xrightarrow{x_2} p_2 \dots \xrightarrow{x_k} p_k$, et il existe de la même façon un calcul réussi de A_2 qui s'écrit $p'_0 \xrightarrow{x_1} p'_1 \xrightarrow{x_2} p'_2 \dots \xrightarrow{x_k} p'_k$.
Alors $(p_0, p'_0) \xrightarrow{x_1} (p_1, p'_1) \xrightarrow{x_2} (p_2, p'_2) \dots \xrightarrow{x_k} (p_k, p'_k)$ est un calcul réussi de A, ce qui termine la démonstration.

Question I.4
On obtient l'automate de la figure 3, page 2, où on a réuni deux flèches étiquetées par a et b par une seule étiquettée par a, b pour simplifier.

Question I.5
On choisit $T' = T$ mais $F' = Q \setminus F$, de sorte que les calculs de A et de A' sont **les mêmes**, mais que quand un calcul est réussi dans un automate, il ne l'est pas dans l'autre.
Bien sûr, cette construction est correcte car les automates construits sont complets, sans quoi un mot qui ne serait pas dans L pourrait bloquer l'automate A !

On en déduit les schémas des deux automates A'_{1_ex} et A'_{2_ex} de la figure 4.

Question I.6
On obtient l'automate A_{5_ex} de la figure 5.

Question I.7
Comme A_{4_ex} reconnaît le langage $M_{ex} = \overline{L_{1\_ex}} \cup \overline{L_{2\_ex}}$, il en est de même pour A_{5_ex}.

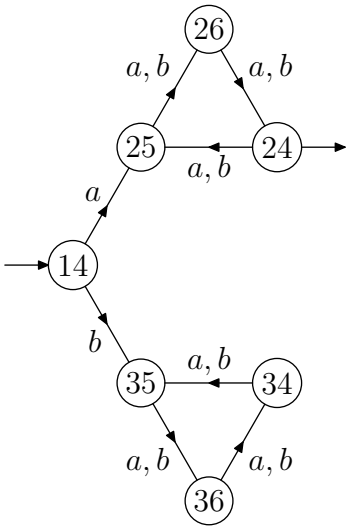

Figure 3 l'automate A_{3_ex}

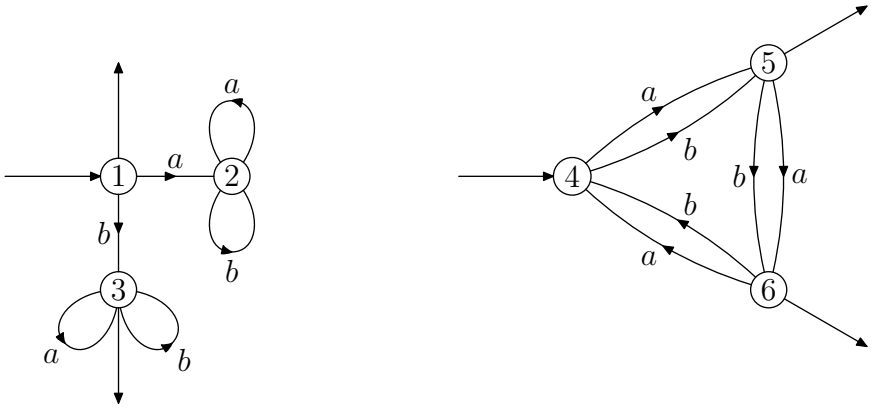

Figure 4 les automates A'_{1_ex} et A'_{2_ex}

Mais le complémentaire de M_{ex} n'est autre que l'intersection $L_{1\_ex} \cap L_{2\_ex}$, et pour obtenir l'automate demandé, il suffit d'intervertir états finals et non finals dans A_{5_ex} (puisqu'il est déterministe complet) : on retrouve ici exactement A_{3_ex}.

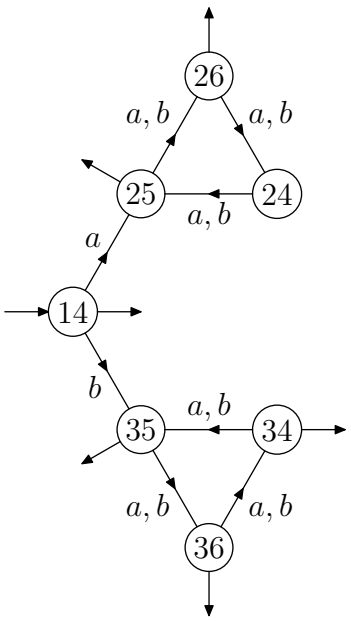

Figure 5 l'automate A_{5_ex}

Partie II — Le problème du voyageur de commerce

Questions préliminaires

Question II.8
Il y a $n!$ permutations de n éléments. Mais deux permutations égales à une rotation près des indices correspondent en réalité au même tour. Pour dénombrer les tours, on décide par exemple de compter les permutations qui commencent avec 0, et il n'y en a plus que $(n-1)!$. Comme de plus le graphe n'est pas orienté, il convient de diviser par 2, le sens de parcours ne changeant pas le tour effectué, et la réponse est donc qu'il y a $\dfrac{(n-1)!}{2}$ tours utiles.
Chaque calcul de poids d'un tour coûte $O(n)$. L'algorithme naïf par exhaustion des tours sera donc de complexité $O\left(\dfrac{(n-1)!}{2} \times n\right) = O(n!)$, ce qui est rédhibitoire.

Question II.9
On écrit facilement le programme 1.

```
1  let poids_tour poids T =
2    let n = vect_length T in
3    let p = ref poids.(T.(n-1)).(T.(0)) in
4      for i = 0 to n - 2 do p := !p + poids.(T.(i)).(T.(i+1)) done ;
5      !p ;;
```

Programme 1 La fonction `poids_tour`

Question II.10
La complexité demandée est bien entendu linéaire.

Question II.11
On écrit facilement le programme 2.

```
6  let plus_proche poids S x =
7    let n = vect_length S in
8    let p = ref 0 and m = ref MAX_POIDS in
9      for i = 0 to n - 1 do
10       if S.(i) = 1 && poids.(i).(x) < !m then ( p:=i ; m:=poids.(i).(x) )
11     done ;
12     !p ;;
```

Programme 2 La fonction `plus_proche`

Question II.12
La complexité demandée est bien entendu $O(n)$ là encore.

L'heuristique du plus proche voisin

Question II.13
Le tour obtenu pour le graphe *G_ex* en partant de 0 est $(0, 1, 2, 3, 4)$, qui est de poids égal à 22.

Question II.14
Le tour obtenu pour le graphe *G_ex* en partant de 1 est $(1, 0, 4, 2, 3)$, qui est de poids égal à 22 lui aussi.

Question II.15
On propose le programme 3. La variable t contient le tour en construction, la variable s code le sous-ensemble des sommets qui ne sont pas encore dans le tour.

```
13  let tour_plus_proche poids depart =
14    let n = vect_length poids.(0) in
15    let t = make_vect n 0 and s = make_vect n 1 in
16      t.(0) <- depart ; s.(depart) <- 0 ;
17      for i = 1 to n - 1 do
18        let p = plus_proche poids s t.(i-1) in
19         t.(i) <- p ; s.(p) <- 0
20      done ;
21      t ;;
```

Programme 3 La fonction `tour_plus_proche`

Remarque : on peut vérifier que les tours obtenus pour le graphe G_ex en partant de 0, 1 ou 3 sont de poids 22, alors que les tours obtenus en partant de 2 ou 4 sont de poids 18 : il s'agit d'ailleurs du même tour $(0, 1, 4, 2, 3)$.

Question II.16
La complexité est $O(n^2)$: il y a en effet $n - 1$ appels à la fonction `plus_proche`.

Question II.17
On vérifiera par exemple que dans le graphe de la figure 6, tous les tours obtenus (quel que soit le sommet de départ) par l'heuristique du plus proche voisin sont de poids 40 : il s'agit de $(0, 1, 3, 2)$, $(1, 3, 2, 0)$, $(2, 3, 1, 0)$ et $(3, 1, 0, 2)$. Pourtant il existe un tour de poids 39 : il s'agit de $(0, 1, 2, 3)$.

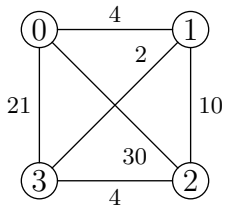

Figure 6 l'heuristique du plus proche voisin en défaut

La méthode par amélioration itérative

Question II.18
On a dessiné le tour T en noir et le tour T' en rouge dans la figure 7. La permutation induisant T' demandée est $(3, 4, 1, 7, 2, 6, 0, 5)$.
On remarquera que la portion entre i' et j' est parcourue à rebours dans le nouveau tour.

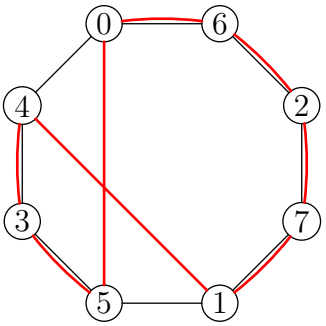

Figure 7 le tour $T' = 2{-}opt(T, 1, 6)$

Question II.19
Le tour T' s'obtient en retournant la tranche des éléments de T d'indices $i + 1$ à j inclus, ce qu'on peut réaliser en $\left\lfloor \frac{j-i}{2} \right\rfloor$ échanges.
On obtient le programme 4, page 5.

```
22  let deux_opt T i j =
23    let échange k l = let x = T.(k) in T.(k) <- T.(l) ; T.(l) <- x
24    in
25    let n = vect_length T in
26      for k = i+1 to (i+j)/2 do échange k (i + j + 1 - k) done ;
27      T ;;
```

Programme 4 La fonction `deux_opt`

Question II.20
Le cas le plus coûteux est celui où la tranche à retourner est la plus longue, donc quand $j-i$ est maximal, donc égal à $n-2$ d'après les conditions imposées. Dans ce cas, la complexité est $O(j-i) = O(n)$.

Question II.21
Considérons le graphe G_ex et le tour $T = (0,1,2,3,4)$, de poids 22. Les deux arêtes les plus chères de ce tour relient 1 et 2 d'une part et 3 et 4 d'autre part. Mais on vérifie que la transformation 2-opt de paramètres 1 et 3 n'améliore pas la situation.
On vérifie plus généralement qu'aucune transformation 2-opt n'arrive à faire diminuer ce poids : l'algorithme ne trouve pas mieux que le tour initial.

Question II.22
Considérons le graphe G_ex et le tour $T = (1,0,4,2,3)$, de poids 22. Les deux arêtes les plus chères de ce tour relient 3 et 1 d'une part et 0 et 4 d'autre part.
On effectue donc $T \leftarrow 2\text{-opt}(T,1,4) = (1,0,3,2,4)$, de poids 18. On ne trouve alors plus de transformation 2-opt qui fasse diminuer ce poids, et l'algorithme retourne ce tour — dont on peut d'ailleurs vérifier à la main qu'il est bien l'unique tour de poids minimal.

Question II.23
Les réponses aux deux questions précédentes montrent que la méthode par amélioration itérative peut trouver ou ne pas trouver un tour de poids minimum, selon le cas.

La méthode par séparation et évaluation

Question II.24
Avec $C\_ex = (0,3,1)$, on a $U = \{2,4\}$, $poids(G\_ex, C\_ex) = 5+12 = 17$, $eval1(G\_ex, C\_ex, 0) = 4$, $eval1(G\_ex, C\_ex, 1) = 6$, puis $eval2(G\_ex, C\_ex, 2) = 3+6 = 9$ et $eval2(G\_ex, C\_ex, 4) = 3+4 = 7$, donc finalement $eval(G\_ex, C\_ex) = 17 + \left\lceil \dfrac{4+6+9+7}{2} \right\rceil = 30$.

Question II.25

Notons $\pi_{i,j} = poids(\{t_i, t_j\})$ et évaluons pour un tour $T = (t_0, t_1, \ldots, t_{n-1})$, $2poids(G,T) = \sum_{i=0}^{n} (\pi_{i-1,i} + \pi_{i,i+1})$, avec la convention que les indices sont calculés modulo n.
Dans cette somme, on retrouve

- ▷ 2 fois le poids de chaque arête de la chaîne C, donc $2poids(G,C)$;
- ▷ le poids a de l'arête qui entre dans la chaîne C ;
- ▷ le poids b de l'arête qui sort de la chaîne C ;
- ▷ la somme c des arêtes qui relient chaque élément x de U à ses voisins, qui sont éléments de $U \cup \{c_0, c_{p-1}\}$, où on a noté U l'ensemble des sommets de T (ou du graphe) qui ne sont pas dans C.

Mais $a \geqslant eval1(G,C,c_0)$ et $b \geqslant eval1(G,C,c_{p-1})$, et, d'autre part, $c \geqslant \sum_{x\in U} eval2(G,C,x)$, de sorte que $2(poids(G,T) - poids(G,C)) = a+b+c \geqslant eval1(G,C,c_0) + eval1(G,C,c_{p-1}) + \sum_{x\in U} eval2(G,C,x)$.
Au moment de diviser par deux, comme $a+b+c$ est nécessairement pair, on peut bien utiliser la notation avec $\lceil \ \ \rceil$ et obtenir ainsi la minoration souhaitée : $poids(G,T) \geqslant eval(G,C)$.

Question II.26
Observons tout d'abord que si la chaîne C vérifie $p = n - 1$, U est réduit à un singleton $\{x\}$ et il existe un unique tour $T = (c_0, \ldots, c_{p-1}, x)$ contenant C, avec $poids(G, T) = eval(G, C)$.
Dans cette question on pose $G = G\_ex$, et on note T_0 le meilleur tour courant, p_0 son poids, et C la chaîne courante :
Au début $C = (0, 3)$, $T_0 = (0, 1, 2, 3, 4)$ et $p_0 = 22$.
On évalue $eval(G, C) = 16 < p_0$, donc on doit considérer successivement :

1. $C = (0, 3, 1)$: on trouve $eval(G, C) = 30 > p_0$, donc on abandonne cette chaîne ;
2. $C = (0, 3, 2)$: on trouve $eval(G, C) = 16 < p_0$, donc on regarde plus avant.
   - $C = (0, 3, 2, 1)$ vérifie $eval(G, C) = poids(G, (0, 3, 2, 1, 4)) = 24 > p_0$ qu'on abandonne ;
   - $C = (0, 3, 2, 4)$ vérifie $eval(G, C) = poids(G, (0, 3, 2, 4, 1)) = 18 < p_0$: c'est le meilleur score, donc on pose $T_0 = (0, 3, 2, 4, 1)$ et $p_0 = 18$;
3. $C = (0, 3, 4)$: on trouve $eval(G, C) = 24 > p_0$, donc on abandonne cette chaîne.

L'algorithme termine avec le tour $T_0 = (0, 3, 2, 4, 1)$ de poids 18.

Question II.27
Il est facile d'écrire le programme 5 : on cherche le sommet y de S le plus proche de x puis le sommet z de $S' = S \setminus \{y\}$ le plus proche de x, ce qui fournit les deux arêtes demandées.

```
28  let somme_deux_plus_légères poids S x =
29    let n = vect_length poids.(0) in
30    let y = plus_proche poids S x in
31    let S' = init_vect n (function i -> S.(i)) in
32      S'.(y) <- 0 ;
33      let z = plus_proche poids S' x
34      in
35        poids.(x).(y) + poids.(x).(z) ;;
```

Programme 5 La fonction `somme_deux_plus_légères`

Question II.28
On écrit le programme 6 de la page 7 de la façon suivante : on note n la taille du graphe, p la taille de la chaîne, `c0` et `cp1` sont ses extrémités. On calcule le code de l'ensemble U des sommets qui ne sont pas dans la chaîne, et U' s'obtient en ajoutant à U les extémités de la chaîne.
On cherche le plus proche voisin x de c_0 dans U et le plus proche voisin y de c_{p-1} dans U, et on initialise la somme `s` à $eval1(G, C, x_0) + eval1(G, C, c_{p-1})$.
Au moment de calculer $eval2(G, C, x)$ pour tout élément $x \in U$, il ne faut pas oublier de temporairement supprimer x de U' avant l'appel de `somme_deux_plus_légères` : c'est la petite gymnastique des lignes 45 et 47.
Enfin, pour tout entier a, on a bien $\lceil a/2 \rceil = \lfloor (a+1)/2 \rfloor$: c'est le calcul qu'on fait en ligne 49.

Question II.29
On écrit une fonction auxiliaire de recopie de vecteur et une autre qui étend un vecteur en lui ajoutant un élément. Le programme proposé n'est pas optimal : il vaudrait mieux garder la liste des éléments de C et le poids du meilleur tour courant en mémoire plutôt que les recalculer à chaque appel récursif. C'est ce qu'on a fait dans la deuxième version proposée, `tour_min2` du programme 7, page 7.

Question II.30
La première stratégie étudiée est peu probante, on l'a vu.
On pourra commencer par appliquer une version simplifiée des transformations 2-opt : on commence avec le tour induit par exemple par la permutation identique, et on applique l'algorithme étudié dans les questions 18 à 23 en choisissant systématiquement les deux arêtes les plus coûteuses.
On obtient ainsi un tour, peut-être pas optimal, mais de qualité déjà significative.
On passe alors à l'algorithme étudié dans cette dernière partie.

```
let eval poids C =
  let n = vect_length poids.(0) and p = vect_length C in
  let c0 = C.(0) and cp1 = C.(p-1) in
  let U = make_vect n 1 and U' = make_vect n 1 in
    for i = 0 to p-1 do U.(C.(i)) <- 0 ; U'.(C.(i)) <- 0 done ;
    U'.(c0) <- 1 ; U'.(cp1) <- 1 ;
    let x = plus_proche poids U c0 and y = plus_proche poids U cp1 in
      let s = ref (poids.(c0).(x)+poids.(cp1).(y)) in
       for i = 0 to n-1 do if U.(i)=1 then begin
         U'.(i) <- 0 ;
         s := !s + somme_deux_plus_légères poids U' i ;
         U'.(i) <- 1
       end done ;
       s := (!s+1)/2 ;
       for i = 0 to p-2 do s := !s + poids.(C.(i)).(C.(i+1)) done ;
       !s ;;
```

Programme 6 La fonction `eval`

```
let recopie_vecteur u v = (* u <- v *)
  for i = 0 to vect_length u - 1 do u.(i) <- v.(i) done ;;

let étend_vecteur u x = (* ajoute x en fin du vecteur u *)
  let p = vect_length u in let v = make_vect (p+1) 0 in
    for i = 0 to p-1 do v.(i) <- u.(i) done ;
    v.(p) <- x ;
    v ;;

let rec tour_min poids C meilleur_tour =
  let n = vect_length poids.(0) and p = vect_length C
  and score = poids_tour poids meilleur_tour in
    if p = n then
      ( if poids_tour poids C < score then recopie_vecteur meilleur_tour C )
    else if eval poids C < score then
      let L = list_of_vect C in
       for x = 0 to n-1 do if not (mem x L) then
         tour_min poids (étend_vecteur C x) meilleur_tour
       done ;;

let tour_min2 poids C meilleur_tour =
  let n = vect_length poids.(0) and score = ref (poids_tour poids meilleur_tour) in
  let rec tour_min_rec C L =
    let p = vect_length C in
      if p = n then
       let s' = poids_tour poids C in
         ( if s' < !score then (score := s' ; recopie_vecteur meilleur_tour C ))
      else if eval poids C < !score then
       for x = 0 to n-1 do if not (mem x L) then
         tour_min_rec (étend_vecteur C x) (x :: L)
       done
  in
    tour_min_rec C (list_of_vect C) ;
    (!score,meilleur_tour) ;;
```

Programme 7 La fonction `tour_min`