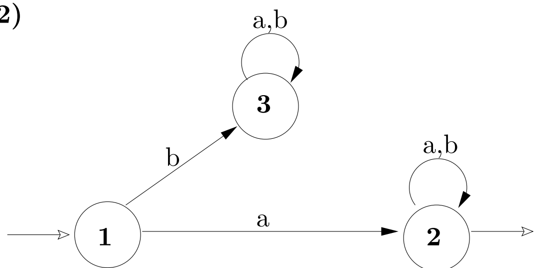
Mines-Ponts 2008. Option Informatique.

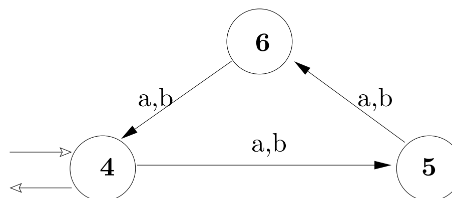
Corrigé pour serveur UPS par JL. Lamard (jean-louis.lamard@prepas.org)

Partie I. Problème sur les automates.

1) et 2)



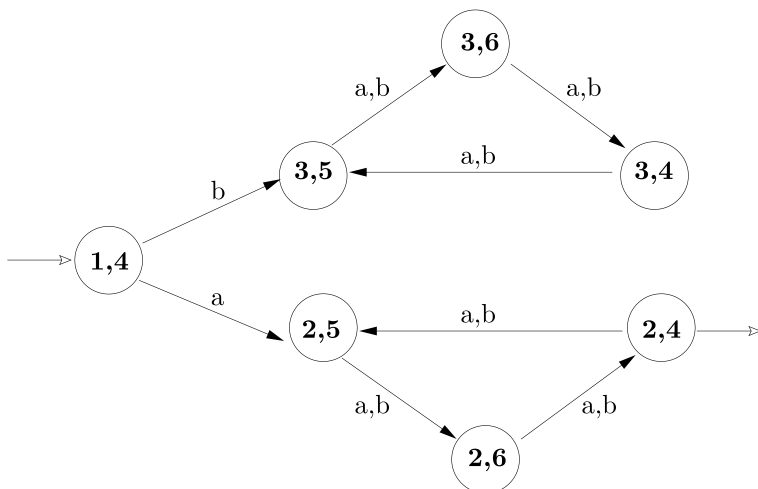
Automate A_{1_ex}



Automate A_{2_ex}

3) Soit l'automate A de l'énoncé défini par les transitions : $(p_1, p_2) \xrightarrow{x} (q_1, q_2)$ avec $p_i \xrightarrow{x} q_i$.
Il est alors immédiat par récurrence sur $n = |m|$ que $(p_1, p_2) \xrightarrow{m} (q_1, q_2)$ si et seulement si $p_i \xrightarrow{m} q_i$ de sorte que le langage reconnu par cet automate est bien $L_1 \cap L_2$.

4)



Automate A_{3_ex}

5. L'automate $A' =< \Sigma, Q, T', \{q_0\}, F' >$ avec $T' = T$ et $F' = \overline{F}$ est déterministe et complet puisque A l'est et le langage reconnu par cet automate est clairement $\overline{L}$ par construction.

La représentation graphique de A'_{1_ex} et A'_{2_ex} est immédiate : il s'agit des automates "complémentaires" respectivement de A_{1_ex} et A_{2_ex} obtenus en supprimant l'unique flèche sortante et en plaçant une à chaque état qui n'en possède pas.

6) Les états accessibles de A'_{1_ex} sont les états 2 et 3. Tous les états de A'_{2_ex} sont accessibles. Outre l'état initial $(1, 4)$, l'automate A_{5_ex} déterministe et complet comporte donc 6 états.
On constate facilement qu'on obtient l'automate complémentaire de l'automate A_{3_ex}.

7) L'automate A_{5_ex} reconnaît (comme l'automate A_{4_ex}) le langage $L \underset{\text{DEF}}{=} \overline{L_{1_ex}} \cup \overline{L_{2_ex}}$.
Son complémentaire, qui n'est autre que l'automate A_{3_ex} compte-tenu de la question précédente, reconnait donc le langage $\overline{L}$ c'est à dire $L_{1_ex} \cap L_{2_ex}$.

Ainsi les deux constructions (par automate produit et par automate complémentaire de l'union des automates complémentaires) fournissent le même automate.

Partie II. Problème du voyageur de commerce.

Première partie : questions préliminaires.

8) Il y a $n!$ permutations différentes mais à un même tour correspondent n permutations différentes (déduites par permutation circulaire) Il existe donc $(n-1)!$ tours différents.
Comme deux tours ne différant que par l'orientation ont même poids, la méthode exhaustive avec $n = 50$ conduirait à calculer le poids de $49!/2 \simeq 0.3 \times 10^{63}$ tours, ce qui est évidemment déraisonnable ! Pour une machine calculant 10^6 poids à la seconde, le temps de calcul serait de l'ordre de 10^{55} années ce qui dépasse plus que largement l'âge de l'univers !

9) Écriture immédiate :

```
let poids_tour poids T =
  let n = vect_length T in let somme = ref poids.(T.(n-1)).(T.(0)) in
    for i = 0 to n-2 do somme := !somme + poids.(T.(i)).(T.(i+1)) done;
    !somme
;;
poids_tour : int vect vect -> int vect -> int = <fun>
```

10) La complexité est bien entendu linéaire par rapport à n : $n-1$ additions.

11) On commence par déterminer le plus petit sommet du sous-ensemble S : initialisation de la variable *min* puis la suite est classique. On notera l'ordre des booléens dans le "et logique" de la boucle "for" pour ne pas faire de comparaison de poids lors de la rencontre d'un sommet n'appartenant pas au sous-ensemble.

```
let plus_proche poids S x =
  if (S.(x) = 1)  then failwith "Le sommet appartient au sous-ensemble !";
  let n = vect_length poids
  and min = ref (let i = ref 0 in (while S.(!i) = 0 do incr i done; !i)) in
    let val_min = ref poids.(x).(!min) in
      for j= (!min + 1) to n - 1 do
        if ((S.(j) = 1) & (poids.(x).(j) < !val_min)) then
          begin min := j; val_min := poids.(x).(j) end
      done;
      !min
;;
plus_proche : 'a vect vect -> int vect -> int -> int = <fun>
```

Remarque : si l'on connait, ce qui est le cas dans l'énoncé, un majorant de l'ensemble des poids, on peut bien sûr écrire une version plus simple :

```
let plus_proche poids S x =
  if (S.(x) = 1)  then failwith "Le sommet appartient au sous-ensemble !";
  let n = vect_length poids and min = ref 0 and val_min = ref MAX_POIDS in
    for i = 0 to n -1 do
      if ((S.(i) = 1) & (poids.(x).(i) < !val_min)) then
        begin min := i; val_min := poids.(x).(i) end
    done;
    !min
;;
plus_proche : int vect vect -> int vect -> int -> int = <fun>
```

12) On effectue n test d'appartenance au sous-ensemble et au plus $p-1$ comparaisons de poids si p est le nombre de points du sous-ensemble. La complexité est donc là encore linéaire.

Seconde partie : l'heuristique du plus proche voisin.

13) et 14)
En partant du sommet 0 on obtient le tour $(0, 1, 2, 3, 4)$ de poids 22 et en partant du sommet 1 le tour $(1, 0, 4, 2, 3)$ de poids 22 également.

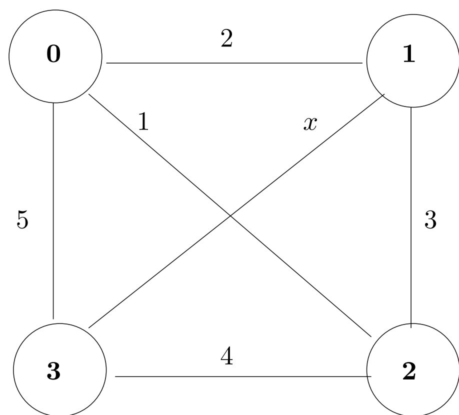
15) Voir le programme très simple de la page suivante qui maintient à jour deux variables : *tour* qui désigne le tour en construction et *ensemble* qui code l'ensemble des points déjà construits.

```
let tour_plus_proche poids depart = let n = vect_length poids in
  let tour = make_vect n depart
  and ensemble = make_vect n 1 in
    ensemble.(depart) <- 0;
    for i = 1 to n-1 do
      let pp = plus_proche poids ensemble tour.(i - 1) in
        tour.(i) <- pp;
        ensemble.(pp) <- 0
    done;
    tour
;;
tour_plus_proche : int vect vect -> int -> int vect = <fun>
```

16) On effectue $n-1$ appels à la fonction `plus_proche` donc $(n-1)n \sim n^2$ tests d'appartenance aux divers sous-ensembles.
Le nombre de comparaisons de poids effectué est $(n-1)+(n-2)+\ldots+2 \sim \dfrac{n^2}{2}$

17) La question est soit mal posée soit sans grand intérêt.

- En effet s'il s'agit de prouver qu'un tour particulier défini par cet algorithme (dans suite on dira un tour "ppv") n'est pas toujours de poids minimum, il suffit d'exhiber un autre tour "ppv" de poids strictement plus petit.
Pour cela nul besoin d'aller chercher un autre graphe que celui de l'exemple de l'énoncé. En effet le tour "ppv" de sommet 2 est $(2,3,0,1,4)$ de poids $19<22$.
- La vraie question est de savoir si le tour "ppv" de poids le plus petit parmi les n tours "ppv" est de poids minimum. La réponse est non.
Pour ne pas tatonner dans l'attribution des poids des arêtes d'un graphe d'ordre 4 (ce qui risque d'être long avant de trouver un contre-exemple !), on procède ainsi : on affecte à l'une des diagonales le plus petit poids soit 1 et à l'autre le plus grand soit x. Ainsi la diagonale de poids fort se trouvera toujours dans un tour "ppv".
Ce qui fournit par exemple le graphe suivant :



Le tour par les côtés $(0,1,2,3)$ a pour poids 14.
Les tours "ppv" qui sont $(0,2,1,3)$, $(1,0,2,3)$, $(2,0,1,3)$ et $(3,2,0,1)$ ont pour poids $9+x$ pour le premier et $7+x$ pour les trois autres.
Ainsi pour $x=99$ les tours "ppv" sont d'un poids très éloigné du poids minimum !

Troisième partie : méthode par amélioration itérative.

18) La transformation consiste ici à retirer les arêtes $(4,0)$ et $(1,5)$ et à ajouter les arêtes ("diagonales" sur le dessin) $(4,1)$ et $(0,5)$.
Ce tour est induit par la permutation $(3,4,1,7,2,6,0,5)$.

19) La permutation induisant le tour T' s'obtient à partir de celle induisant le tour T en "inversant" le sens de parcours des sommets d'indices compris entre $i+1$ et j. Ce qui s'obtient par $\mathrm{E}\left(\dfrac{j-i}{2}\right)$ échanges.

D'où le programme suivant :

```
let deux_opt T i j =
  let nb = (j - i) /2 in
    for k = 0 to nb -1 do echange T (i + 1 + k) (j - k) done
      where echange v i j = let temp = v.(i) in
        v.(i) <- v.(j);
        v.(j) <- temp
;;
deux_opt : 'a vect -> int -> int -> unit = <fun>
```

20) La complexité est de $\mathrm{E}\left(\dfrac{j-i}{2}\right)$ échanges (un échange valant 3 affectations) comme on vient de le voir. La complexité maximale est obtenue lorsque l'écart $j-i$ est maximum c'est à dire égal à $n-2$ (vu les conditions sur i et j). Ainsi la complexité maximale est-elle $O(n)$.

21), 22) et 23)

- Pour le graphe G_ex on a $n=5$ donc les seules valeurs possibles de i sont 0, 1 et 2.
  Si $i=2$ alors nécessairement $j=4$.
  Si $i=1$ alors $j=3$ ou $j=4$.
  Si $i=0$ alors $j=2$ ou $j=3$ (car le couple $(i,j)=(0,4)$ est exclu).
- Lorsque $T=(0,1,2,3,4)$ de poids 22 il vient :

  $2_opt(T,2,4)=T$ de poids 22;
  $2_opt(T,1,3)=(0,1,3,2,4)$ de poids 22;
  $2_opt(T,1,4)=(0,1,4,3,2)$ de poids 29;
  $2_opt(T,0,2)=(0,2,1,3,4)$ de poids 37;
  $2_opt(T,0,3)=(0,3,2,1,4)$ de poids 28.

  Ainsi dans ce cas la méthode itérative ne permet pas de progresser.
- Lorsque $T=(1,0,4,2,3)$ de poids 22 on obtient :

  $2_opt(T,2,4)=T$ de poids 22;
  $2_opt(T,1,3)=(1,0,2,4,3)$ de poids 28;
  $2_opt(T,1,4)=(1,0,3,2,4)$ de poids 18;
  $2_opt(T,0,2)=(1,4,0,2,3)$ de poids 35;
  $2_opt(T,0,3)=(1,2,4,0,3)$ de poids 30.

  On a ainsi obtenu un tour de poids plus faible : $T'=(1,0,3,2,4)$.

  En recommençant comme ci-dessus à partir de T', on constaterait qu'on ne progresse plus. C'est donc le résultat de l'algorithme.
  Rien ne prouve d'ailleurs que T' soit un tour de poids minimum ! On pourrait vérifier "à la main" que c'est bien le cas ici.
- Le premier exemple montre que cette méthode ne permet pas toujours de trouver un tour de poids minimum.
- REMARQUE : l'heuristique, telle qu'elle est décrite dans l'énoncé, conduit peu ou prou à envisager toutes les transformations possibles ou tout au moins jusqu'à en trouver éventuellement une fournissant un tour de poids plus faible. D'où éventuellement un grand nombre de tentatives avec cet algorithme qui, comme on vient de le voir, ne permet pas toujours de progresser même s'il existe un tour de poids plus faible !
  Il paraît plus raisonnable d'essayer seulement d'éliminer les deux arêtes de poids le plus élevé et donc de n'essayer qu'une transformation bien que rien n'assure que cette transformation soit la meilleure possible.

  Pour le premier exemple $T=(0,1,2,3,4)$ il s'agit des arêtes $(3,4)$ et $(1,2)$ ce qui correspond au couple $(i,j)=(1,3)$ ce qui fournit un tour de même poids.

  Pour le second exemple $T=(1,0,4,2,3)$ il s'agit des arêtes $(3,1)$ et $(0,4)$ ce qui correspond au couple $(i,j)=(1,4)$ ce qui fournit bien le tour $(1,0,3,2,4)$ de poids 18.

Quatrième partie : méthode par séparation et évaluation.

24) Avec $G=G_{ex}$ et $C=C_{ex}=(0,3,1)$, il vient $eval1(G,C,0)=4$, $eval1(G,C,1)=6$, $eval2(G,C,2)=9$ et $eval2(G,C,4)=7$ d'où $eval(G,C)=poids(G,C)+13=17+13=30$.

25) Écrivons $T=(c_0,c_1,\ldots,c_{p-1},t_p,\ldots,t_{n-1})$ de sorte que en notant $\ell(a,b)$ la longueur d'une arête (a,b) :
$2\times poids(G,T)=2\times poids(G,C)+\ell(c_{p-1},t_p)+\ell(t_{n-1},c_0)+A$ avec
$A=\Big(\ell(c_{p-1},t_p)+\ell(t_p,t_{p+1})\Big)+\Big(\ell(t_p,t_{p+1})+\ell(t_{p+1},t_{p+2})\Big)+\cdots+\Big(\ell(t_{n-2},t_{n-1})+\ell(t_{n-1},c_0)\Big)$.

Or $\ell(c_{p-1},t_p)\geqslant eval1(G,C,c_{p-1})$ et $\ell(t_{n-1},c_0)\geqslant eval1(G,C,c_0)$ d'une part et par ailleurs :

$\ell(c_{p-1},t_p)+\ell(t_p,t_{p+1})\geqslant eval2(G,C,t_p)$
$\ell(t_p,t_{p+1})+\ell(t_{p+1},t_{p+2})\geqslant eval2(G,C,t_{p+1})$
$\ldots$
$\ell(t_{n-2},t_{n-1})+\ell(t_{n-1},c_0)\geqslant eval2(G,C,t_{n-1})$

Ainsi $A\geqslant \sum\limits_{x\in U} eval2(G,C,x)$

Donc $2\Big(poids(G,T) - poids(G,C)\Big) \geqslant B$ avec $B = eval1(G,C,c_0) + eval1(G,C,c_{p-1}) + \sum_{x \in U} eval2(G,C,x)$.
Ainsi $poids(G,T) - poids(G,C) \geqslant \dfrac{1}{2}B$.
La fonction partie entière par excès étant croissante, il vient : $\lceil poids(G,T) - poids(G,C) \rceil \geqslant \left\lceil \dfrac{1}{2}B \right\rceil$
soit $poids(G,T) - poids(G,C) \geqslant \left\lceil \dfrac{1}{2}B \right\rceil$ ce qui est bien l'inégalité demandée.

26) Dans l'algorithme décrit dans cette question, on va au maximum jusqu'à $p = n-1$ (et non $p = n$) car alors U est réduit à un singleton et il n'existe qu'un seul tour prolongeant la chaîne dont il suffit de comparer le poids à celui de *meilleur_tour*.

On notera C la chaîne courante, T_{min} le meilleur tour courant et p_{min} son poids.

- Initialisation : $C = (0,3)$, $T_{min} = (0,1,2,3,4)$ et $p_{min} = 22$.
- $eval(G,C) = 16 < p_{min}$ donc :
  - $C \longleftarrow (0,3,1) \Longrightarrow eval(G,C) = 30 > p_{min}$ donc fin pour cette chaîne.
  - $C \longleftarrow (0,3,2) \Longrightarrow eval(G,C) = 16 < p_{min}$ donc :
    - $C \longleftarrow (0,3,2,1) \Longrightarrow eval(G,C) = poids(G,(0,3,2,1,4)) = 24 > p_{min}$ donc fin pour cette chaîne.
    - $C \longleftarrow (0,3,2,4) \Longrightarrow eval(G,C) = poids(G,(0,3,2,4,1)) = 18 < p_{min}$ donc $T_{min} \longleftarrow (0,3,2,4,1)$
      $p_{min} \longleftarrow 18$
  - $C \longleftarrow (0,3,4) \Longrightarrow eval(G,C) = 24 > p_{min}$ donc fin pour cette chaîne.
- L'algorithme est terminé : $T_{min} = (0,3,2,4,1)$ et $p_{min} = 18$.

27) L'écriture est immédiate à l'aide de la fonction `plus_proche`.
On notera la remise à sa valeur initiale de `S.(x1)` afin de ne pas avoir un effet de bord modifiant la valeur de `S`.

```
let somme_deux_plus_legeres poids S x =
  let x1 = plus_proche poids S x in
    S.(x1) <- 0;
    let x2 = plus_proche poids S x in
      S.(x1) <- 1;
      poids.(x).(x1) + poids.(x).(x2)
;;
somme_deux_plus_legeres : int vect vect -> int vect -> int -> int = <fun>
```

28) On procède de la manière suivante :
1/ on commence par créer le tableau caractéristique `S` du complémentaire U de la chaîne `C`.
2/ on initialise la variable `result` à $eval1(G,C,c_0) + eval1(G,C,c_{p-1})$
3/ on "rajoute" les points c_0 et c_{p-1} dans le tableau `S`.
4/ on ajoute $\sum_{x \in U} eval2(G,C,x)$ à l'aide d'une boucle `for` à `result`.
(en veillant à ne pas modifier le tableau `S` à chaque tour de boucle).
5/ on transforme `result` en sa partie entière par excès (pour tout entier n on a $\lceil n \rceil = \lfloor (n+1)/2 \rfloor$).
6/ Reste à ajouter $poids(G,C)$ à `result`.

Voir le programme Caml page suivante.

29) On écrit facilement cette fonction qui suit exactement la description de l'algorithme.

Voir le programme page suivante.

30) On peut envisager la méthode suivante bien que rien n'assure que cela fournisse un tour minimum.

1/ on commence par déterminer par la stratégie du plus proche voisin le tour de départ 0 par exemple.
2/ puis on applique la stratégie $2 - opt$ à ce tour en éliminant les 2 arêtes les plus lourdes tant qu'on améliore le score.
3/ on applique alors l'algorithme `tour_min` que l'on vient de voir avec comme argument `meilleur_tour` de départ le tour obtenu en fin d'étape 2 et comme chaîne de départ la chaîne réduite au premier point de ce tour.

Avec le graphe G_{ex} l'étape 1 fournit le tour $(0,1,2,3,4)$ de poids 22.
L'étape 2 ne permet pas de progresser.
L'étape 3 qui consiste en `tour_min` appliqué à la chaîne (0) et au tour $(0,1,2,3,4)$ fournit le tour final $(0,1,4,2,3)$ de poids 18 qui est bien minimum ici.

On peut remarquer que cet exemple n'est pas très significatif car en partant du tour extérieur $(0,1,2,3,4)$ on obtient toujours un tour de poids minimum quelque soit la chaîne de départ réduite à un somment quelconque.

```
let eval poids C =
  let n = vect_length poids.(0)
  and p = vect_length C in
    let S = make_vect n 1 in
      for i = 0 to p-1 do S.(C.(i)) <- 0 done;
      let result = ref ( poids.(C.(0)).(plus_proche poids S (C.(0))) +
                         poids.(C.(p-1)).(plus_proche poids S C.(p-1))) in
        S.(C.(0)) <- 1;
        S.(C.(p-1)) <- 1;
        for i = 0 to n-1 do
          if (i <> C.(0)) & (i <> C.(p-1)) & (S.(i) = 1) then
            begin
              S.(i) <- 0;
              result := !result + somme_deux_plus_legeres poids S i;
              S.(i) <- 1
            end
        done;
        result := (!result + 1) /2;
        for i = 0 to p - 2 do
          result := !result + poids.(C.(i)).(C.(i+1))
        done;
        ! result
;;
eval : int vect vect -> int vect -> int = <fun>
```

```
let rec tour_min poids C meilleur_tour =
  let n = vect_length poids.(0) and p = vect_length C in match n - p with
    | 0  -> if (poids_tour poids C < poids_tour poids meilleur_tour) then
              for i=0 to n-1 do meilleur_tour.(i) <- C.(i) done;
    | _  -> match ((eval poids C) - (poids_tour poids meilleur_tour) < 0) with
      | false -> ()
      | _ -> let S = make_vect n 1 in
               for i = 0 to p-1 do S.(C.(i)) <- 0 done;
               for i = 0 to n-1 do
                 if (S.(i) = 1) then let new_C = make_vect (p+1) i in
                   for j = 0 to p-1 do new_C.(j) <- C.(j) done;
                   tour_min poids new_C meilleur_tour
               done
;;
tour_min : int vect vect -> int vect -> int vect -> unit = <fun>
```

FIN