Chapitre 11

Corrigés des exercices

Corrigés des exercices de la section 5 page 25

Solution de l'énoncé 8 page 12 : Voici les résultats :

— Le vocabulaire choisi :
  — P : il pleut
  — V : il vente
  — PP : je prends mon parapluie
  — PC : je prends mon chapeau
— les formules :
  — $P \wedge V$
  — $P \rightarrow PP$
  — $(\neg P \wedge \neg V) \rightarrow PC$

Solution de l'énoncé 9 page 12 : Voici les résultats :

— Le vocabulaire choisi :
  — C : canari
  — O : oiseau
  — V : vole
  — M : manchot
— les formules :
  — C
  — $C \rightarrow O$
  — $O \rightarrow V$
  — M
  — $M \rightarrow O$
  — $M \rightarrow \neg V$

Solution de l'énoncé 10 page 12 : Voici les résultats :

1. $((A \wedge B) \wedge C) \leftrightarrow (A \wedge (B \wedge C))$, OK (associativité du $\wedge$)
2. $(A \wedge B) \leftrightarrow (B \wedge A)$, OK (commutativité du $\wedge$)
3. $(A \wedge A) \leftrightarrow A$, OK (idempotence du $\wedge$)
4. $(A \wedge (A \vee B)) \leftrightarrow A$, OK
5. $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$, OK (distributivité)
6. $(A \wedge \neg A) \leftrightarrow \bot$, OK
7. $((A \vee B) \vee C) \leftrightarrow (A \vee (B \vee C))$, OK (associativité du $\vee$)
8. $(A \vee B) \leftrightarrow (B \vee A)$, OK (commutativité du $\vee$)

9. $(A \vee A) \leftrightarrow A$, OK (idempotence du $\vee$)
10. $(A \wedge (A \vee B)) \leftrightarrow (A \wedge B)$, NOK
11. $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$, OK (distributivité)
12. $(A \vee \neg A) \leftrightarrow \top$, OK.

Solution de l'énoncé 11 page 13 :

$$\begin{aligned} \Phi = \quad & P \\ \wedge \quad & V \\ \wedge \quad & P \rightarrow PP \\ \wedge \quad & (\neg P \wedge \neg V) \rightarrow PC \end{aligned}$$

Cette formule est satisfiable et, dans ses modèles, P, V, PP sont toujours à `vrai`. Pour la variable PC, on a le choix : la formule $(\neg P \wedge \neg V) \rightarrow PC$ est `vrai` parce que P, V sont `vrai`, quelle que soit la valeur prise pour PC.
Comme PP est `vrai`, la formule $PP \vee PC$ est satisfiable. Donc $\Phi \models (PP \vee PC)$.
Par contre, il existe un modèle de Φ qui falsifie $PP \rightarrow PC$: celle où PC est mis à `faux`. Donc $\Phi \not\models (PP \rightarrow PC)$.

Solution de l'énoncé 12 page 13 :

$$\begin{aligned} \Phi = \quad & C \\ \wedge \quad & C \rightarrow O \\ \wedge \quad & O \rightarrow V \end{aligned}$$

Cette formule est satisfiable et, dans ses modèles, C, O et V sont forcément à `vrai`.

Solution de l'énoncé 13 page 13 :

$$\begin{aligned} \Phi = \quad & C \\ \wedge \quad & C \rightarrow O \\ \wedge \quad & O \rightarrow V \\ \wedge \quad & M \\ \wedge \quad & M \rightarrow O \\ \wedge \quad & M \rightarrow \neg V \end{aligned}$$

Cette formule n'est pas satisfiable (impossible de trouver un modèle) : C, M, O sont forcément à `vrai` mais du coup on a V qui doit être à la fois `vrai` et `faux` ce qui est impossible.

Corrigés des exercices de la section 4.1.2 page 18

Solution de l'énoncé 14 page 18 : Coût total $T(n) = 3 \times T_I(n)$ avec I une affectation qui est une opération élémentaire (donc $T_I(n) = 1$, $\forall n$).
Donc $T(n) = 3 = O(1)$.

Solution de l'énoncé 15 page 18 : Coût total $T(n) = T_C(n) + max(T_{I_1}(n), T_{I_2}(n))$ avec C le test d'une condition et I_i des affectations qui sont des opérations élémentaires.
Donc $T(n) = 1 + max(1, 1) = 2 = O(1)$.

Solution de l'énoncé 16 page 18 : Ici, dans le pire des cas, on va passer n fois dans la boucle. En considérant que le coût des affectations et des tests est 1, on a :
Coût total $T(n) = 1 + \Sigma_{i=1}^{n}(2 + 1) + 2 = 3 + 3n = O(n)$

Solution de l'énoncé 17 page 18 : En considérant que le coût des affectations et des tests est 1, on a :
$T(n) = 2 + \Sigma_{i=1}^{n}(3 + max(1, 0)) + 1 = 3 + 4n = O(n)$.

Solution de l'énoncé 18 page 18 : En considérant que le coût des affectations et des tests est 1, et que celui de la permutation est C_{perm}, on a [1] :
$T(n) \approx C_{perm} \times \Sigma_{i=1}^{n-1} i \approx \frac{C_{perm} \times n \times (n-1)}{2} = O(n^2)$.

Solution de l'énoncé 19 page 18 : En considérant que le coût des affectations et des tests est 1, on a :
$T(n) = 1 + \Sigma_{i=1}^{n-1}(5 + 7(n-i)) + 1 = 1 + 5(n-1) + 7 \times \Sigma_{i=1}^{n-1}(n_i) = 1 + 5(n-1) + \frac{7 \times n \times (n-1)}{2}$
Donc $T(n) = O(n^2)$.

Solution de l'énoncé 20 page 19 : En considérant que le coût des affectations et des tests est 1, on a :
— pour le programme 1, $T(n) = O(n^2)$
— pour le programme 2, $T(n) = O(n)$
Et donc le programme 2 est plus efficace que le programme 1.

1. On va exploiter le fait que $\Sigma_{i=1}^{n} i = \Sigma_{i=1}^{n}(n - i + 1) = \frac{n \times (n-1)}{2}$.

Corrigés des exercices de la section 6.2 page 30

Solution de l'énoncé 2 page 4 : Voici un codage compact des états [2] :

— L'état courant sera représenté par un triplet (CG, MG, B) où CG (resp. MG) dénote le nombre de cannibales (resp. de missionnaires) sur la rive gauche, et B dénote la situation de la barque (rive gauche ou rive droite).

— Etat initial : $(0, 0, D)$ (tous sont sur la rive droite, la barque aussi)

— Etat-but : $(3, 3, G)$ (tous sont sur la rive gauche)

— Un opérateur correspond à un transfert de 1 ou 2 personnes en respectant la contrainte sur le nombre de missionnaires et de cannibales sur chaque rive. La contrainte peut s'exprimer par :

$$\left|\begin{array}{c} (MG \geq CG) \text{ ou } (MG = 0) \text{ (contrainte sur la rive gauche)} \\ \text{et} \\ ((3 - MG) \geq (3 - CG)) \text{ ou } ((3 - MG) = 0) \text{ (contrainte sur la rive droite)} \end{array}\right.$$

La contrainte sur la rive droite peut s'écrire :

$$(MG \leq CG) \text{ ou } (MG = 3) \text{ (contrainte sur la rive droite)}$$

Du coup, la contrainte globale peut se ramener à :

$$\left| \; (MG = CG) \text{ ou } (MG = 0) \text{ ou } (MG = 3).\right.$$

La définition précise de l'opérateur est la suivante :

$$\begin{array}{l} \textsc{Traverser}(CG, MG, B) \quad = \quad \{(CG', MG', B') \text{ tels que} \\ \quad ((B = G) \rightarrow \\ \qquad ((B' = D) \wedge \\ \qquad \exists c, m((0 < c + m \leq 2) \wedge (CG' = CG - c) \wedge (MG' = MG - m)))) \\ \quad \wedge((B = D) \rightarrow \\ \qquad ((B' = G) \wedge \\ \qquad \exists c, m((0 < c + m \leq 2) \wedge (CG' = CG + c) \wedge (MG' = MG + m)))) \\ \quad \wedge(0 \leq CG' \leq 3) \wedge (0 \leq MG' \leq 3) \\ \quad \wedge((MG' = CG') \vee (MG' = 0) \vee (MG' = 3))\} \end{array}$$

L'espace d'états contient 32 états ($4 \times 4 \times 2$) mais il y en a plein qui sont interdits (par exemple $(3, 1, G)$). En réalité le nb d'états autorisé est :

— ceux correspondants à $MG = CG$, excepté si $MG = 0$ ou $MG = 3$ (ce qui couvre les cas où il n'y a personne du côté où est le bateau) : $2 \times 2 = 4$

— ceux correspondants à $MG = 0$, excepté le cas où il n'y a personne du côté où est le bateau : $(4 \times 2) - 1 = 7$

— ceux correspondants à $MG = 3$, excepté le cas où il n'y a personne du côté où est le bateau : $(4 \times 2) - 1 = 7$

Donc, 18 états en tout. Toutefois parmi ces états, il en reste deux qui ne seront pas accessibles :

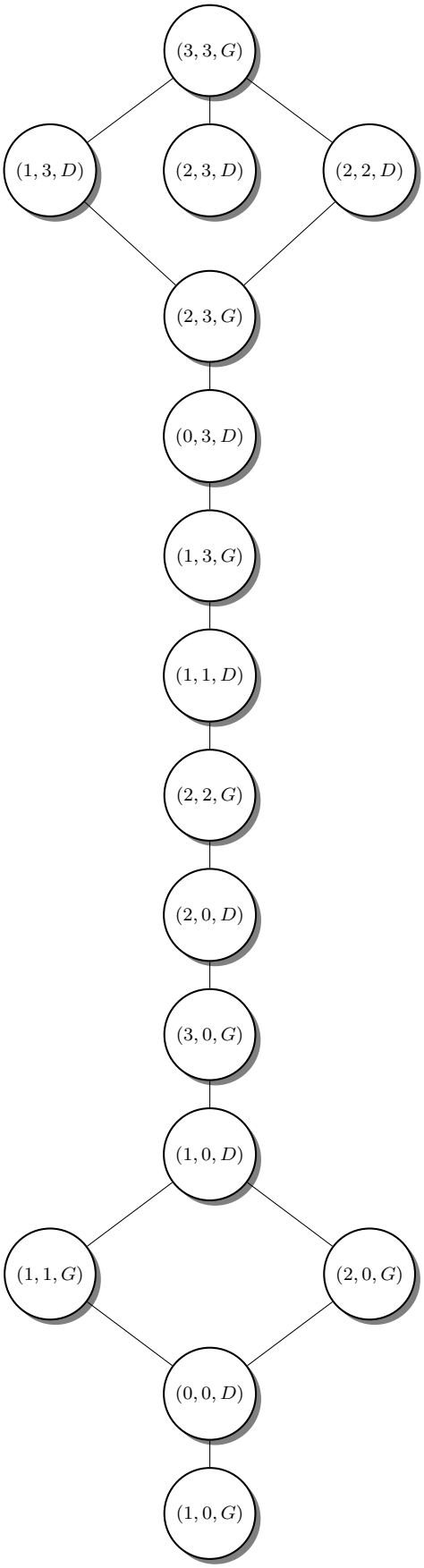
— l'état $(0, 3, G)$ n'est pas accessible car il aurait fallu partir soit de $(0, 1, D)$, soit de $(0, 2, D)$ qui sont deux états interdits ;

— l'état $(3, 0, D)$ n'est pas accessible car il aurait fallu partir soit de $(1, 0, G)$, soit de $(2, 0, G)$ qui sont deux états interdits.

Il reste donc 16 états que l'on va retrouver dans l'espace de recherche. L'espace de recherche peut être représenté par un arbre (avec duplication des états déjà rencontrés) ou un graphe (dans ce cas, si un état a déjà été rencontré on ajoute simplement un arc au graphe). Noter que dans ce problème c'est la séquence des opérateurs qui nous intéresse (chemin menant à un état-but).

L'espace de recherche est :

2. On peut proposer d'autres codages. Par exemple avec la liste des présents sur chaque rive (M, C ou B). L'état initial sera alors représenté par $rg = ()$, $rd = (M, M, M, C, C, C, B)$.


(3, 3, G)
(1, 3, D)
(2, 3, D)
(2, 2, D)
(2, 3, G)
(0, 3, D)
(1, 3, G)
(1, 1, D)
(2, 2, G)
(2, 0, D)
(3, 0, G)
(1, 0, D)
(1, 1, G)
(2, 0, G)
(0, 0, D)
(1, 0, G)

Solution de l'énoncé 3 page 4 : Un état peut être codé sous la forme d'une matrice (exactement la matrice des jetons ds le taquin), les cases de la matrice étant notées $e_{x,y}$ avec x la position en X (sur l'axe horizontal) et y la position en Y (sur l'axe vertical) de la case vide en considérant que la case en bas à gauche du taquin est en position $(0, 0)$.

Par exemple, dans l'état initial, le jeton 4 est dans la case $e_{2,1}$ et la "case vide" est en $e_{1,0}$.

En ce qui concerne les opérateurs, une analyse du problème permet de proposer un codage équivalent et plus compact des opérateurs en considérant que c'est la case vide qui se déplace (et non un jeton)[3]. On obtient donc 4 mouvements de la case vide : "vers le haut", "vers le bas", "vers la gauche", "vers la droite".

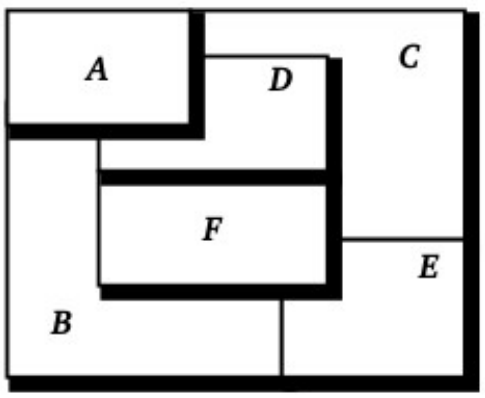
Par exemple, l'opérateur VersLeHaut appliqué à un état e pourrait être défini de la manière suivante (on disposera des accesseurs GetXVide() et GetYVide() renvoyant respectivement la position en X et en Y de la case vide, ainsi que d'une fonction Echange(e,x_1,y_1,x_2,y_2) qui echange le contenu des cases e_{x_1,y_1} et e_{x_2,y_2}) :

VersLeHaut(e) =
  rien, si GetYVide() = 2
  Echange(e,GetXVide(),GetYVide(),GetXVide(),GetYVide()+1), sinon

Dans ce problème, c'est encore la séquence des opérateurs qui nous intéresse si elle existe (chemin menant à un état but) puisqu'on connaît explicitement l'état-but.

Solution de l'énoncé 1 page 4 : Un problème équivalent est la coloration du graphe planaire associé à la carte (on associe un sommet à chaque région et une arête à toute paire de sommets correspondant à des régions adjacentes) :

> Etant donné un graphe G, il s'agit de colorer les sommets de G (en associant à tout sommet une couleur ou un élément de l'ensemble d'indices des couleurs $\{1, 2, \ldots, n\}$) de telle sorte que deux sommets adjacents n'aient pas la même couleur (où n est le nombre de sommets du graphe).



Solution de l'énoncé 4 page 4 : Le codage proposé peut être :

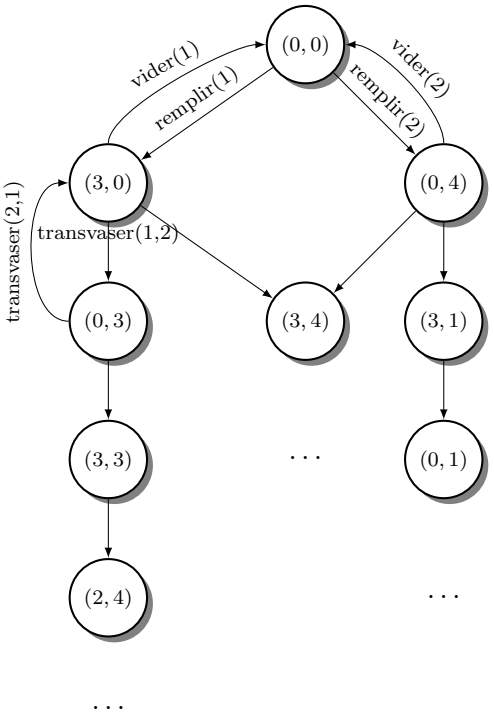
— un état est un doublon donnant le volume du seau 1 et celui du seau 2. Et on a aussi la capacité max de chaque seau qui est définie une fois pour toute ($Max1$ et $Max2$). On a donc une contrainte sur un état $e = (x, y)$: $0 \leq x \leq Max1$ et $0 \leq y \leq Max2$.

— L'état initial est $(0, 0)$

3. Un opérateur est donc une abstraction d'un ensemble d'actions réelles.

— Un état but est soit $(-, 2)$, soit $(2, -)$ avec $-$ représentant n'importe quel volume.
— les opérateurs sont :
  — `remplir(numSeau)` avec la contrainte que le seau `numSeau` n'est pas plein. Soit $e = (x, y)$ l'état courant, on a alors `remplir(1)` permet d'obtenir $(Max1, y)$ et `remplir(2)` permet d'obtenir $(x, Max2)$.
  — `vider(numSeau)` avec la contrainte que le seau `numSeau` n'est pas vide. Soit $e = (x, y)$ l'état courant, on a alors `vider(1)` permet d'obtenir $(0, y)$ et `vider(2)` permet d'obtenir $(x, 0)$.
  — `transvaser(numSeauDepart, numSeauArrivee)` avec la contrainte que le seau `numSeauArrivee` n'est pas plein et que le seau `numSeauDepart` n'est pas vide. Soit e l'état courant avec x dans le seau i et y dans le seau j, on a alors `transvaser(i, j)` permet d'obtenir 0 dans i et $y + x$ dans j si $Max_j \geq y + x$ et sinon $x-(Max_j-y)$ dans i et Max_j dans j (Max_j étant la capacité maximale du seau j).

Avec les valeurs proposées sur l'exemple, cela donne l'espace d'état suivant (on ne dessine qu'une partie du graphe en partant de l'état initial et on ne donne pas non plus tous les labels) :



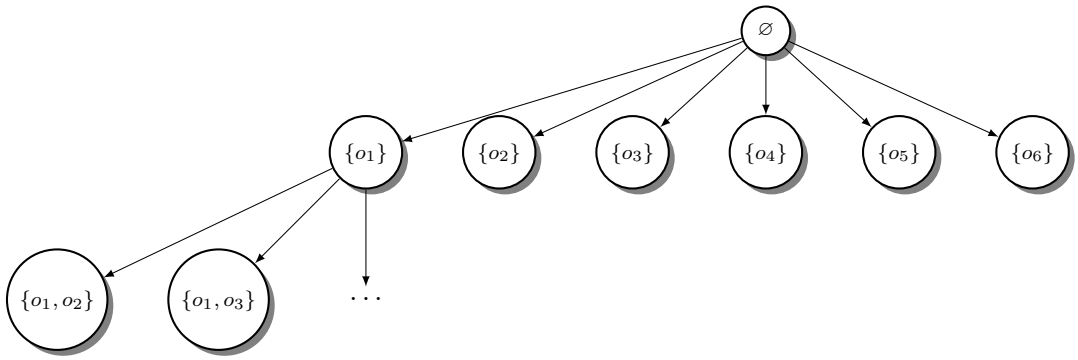
Solution de l'énoncé 5 page 4 : Un codage possible est :

— Un état e sera un ensemble d'objets (ceux qui sont dans le sac). Le poids d'un état e, noté $p(e)$, est la somme des poids des objets dans le sac ; la valeur d'un état e, notée $v(e)$, est la somme des valeurs des objets dans le sac. On a une contrainte à respecter : le poids de e noté $p(e)$ doit être $\leq$ au max (ici 23).
— L'état initial est $e = \varnothing$.
— les états buts dépendent du problème.
  Si on cherche à répondre à la question "Existe-t-il un moyen de remplir complétement le sac ?" (problème de décision), un état e sera un état but ssi $p(e) =$ au max (ici 23).

Si on cherche à répondre à la question "Comment remplir le sac en maximisant la valeur du contenu ?" (problème d'optimisation), un état e sera un état but ssi $p(e) \leq$ au max (ici 23) et $v(e)$ est la plus grande possible.

— L'opérateur permettant de passer d'un état à un autre est `ajouterObjetDsSac(objet)`. Soit e l'état courant, l'application de `ajouterObjetDsSac(`o`)` sur e produira le nouvel état $e' = e \cup \{o\}$. Les contraintes sont que $p(e')$ doit être $\leq$ au max et bien-sûr que o n'appartient pas déjà à e.

Avec les valeurs proposées sur l'exemple, cela donne l'espace d'état suivant (on ne dessine qu'une partie du graphe en partant de l'état initial) :

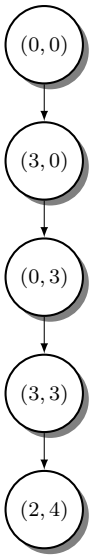


Un exemple d'état interdit : $e = \{o_1, o_2, o_4, o_6\}$ car $p(e) = 25$.

Corrigés des exercices de la section 7.4 page 40

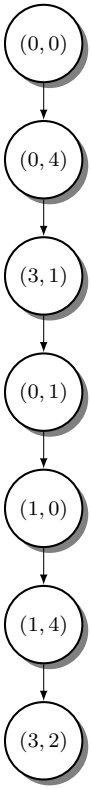
Solution de l'énoncé 21 page 40 : Le résultat de l'algorithme en profondeur d'abord dépend bien-sûr de l'ordre dans lequel on rencontre les fils.

Si le fils $(3,0)$ de $(0,0)$ est developpé avant le fils $(0,4)$, on obtient :



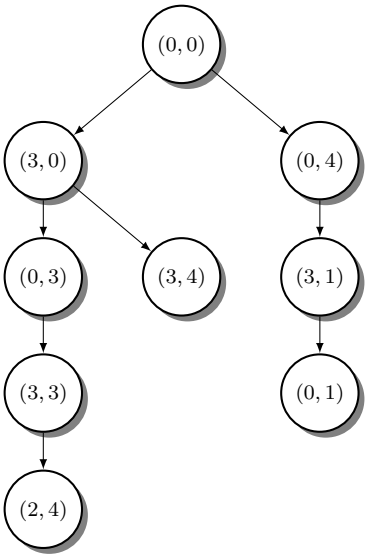
(5 états générés et 4 développés).

Si le fils $(0,4)$ de $(0,0)$ est developpé avant le fils $(3,0)$, on obtient :



(7 états générés et 6 développés).

Avec l'algorithme en largeur d'abord, on a :



(9 états générés et 7 développés).

Solution de l'énoncé 22 page 40 : Une heuristique h pour le problème de décision

du sac à dos pourrait être : $h(e) = Max - p(e)$.

Cette heuristique est coïncidente : si e est un état but $p(e) = Max$ et donc $h(e) = 0$.

Cette heuristique est monotone : soit e' fils de e obtenu par l'ajout de l'objet o ; $p(e') = p(e) + p(o)$. Donc $h(e') = Max - p(e') = Max - (p(e) + p(o)) = h(e) - p(o)$. On a donc bien $h(e) \leq h(e') + \text{coût}(e, e')$.

Puisque h est coïncidente et monotone, on peut alors appliquer la propriété 1 page 39 et conclure que h est minorante. Puis avec la propriété 2 page 39, on conclut que l'algo A* utilisant h est admissible.

Par contre, cette heuristique n'est pas du tout discriminante car $\forall e\ f(e) = Max$! ! Donc on se retrouve finalement à faire un parcours non informé puisque l'information permettant de choisir n'est pas suffisamment discriminante !

Remarquons que les choses sont beaucoup moins simples pour le cas du problème d'optimisation. En plus, dans l'exemple du sac à dos, il s'agit d'un problème de maximisation. Nous n'irons pas plus loin sur ce sujet.

Solution de l'énoncé 23 page 40 :

1. Méthode "coût uniforme". On ne va prendre en compte que les coûts sur les arcs (et pas la fonction h). L'évolution des listes EnAttente et Vus au cours de l'algo est la suivante (entre parenthèses est donné le coût de chaque état = somme des arcs depuis l'état initial jusqu'à l'état courant) :

EnAttente	Vus	e
$\{s_0(0)\}$	$\{\}$	
$\{\}$	$\{s_0\}$	s_0
$\{s_1(1), s_2(1), s_3(3)\}$	$\{s_0\}$	s_0
$\{s_2(1), s_3(3)\}$	$\{s_0, s_1\}$	s_1
$\{s_2(1), s_3(3), s_5(3), s_4(4)\}$	$\{s_0, s_1\}$	s_1
$\{s_3(3), s_5(3), s_4(4)\}$	$\{s_0, s_1, s_2\}$	s_2
$\{s_3(3), s_5(3), s_4(3)\}$	$\{s_0, s_1, s_2\}$	s_2
$\{s_5(3), s_4(3)\}$	$\{s_0, s_1, s_2, s_3\}$	s_3
$\{s_5(3), s_4(3), s_6(4)\}$	$\{s_0, s_1, s_2, s_3\}$	s_3
$\{s_4(3), s_6(4)\}$	$\{s_0, s_1, s_2, s_3, s_5\}$	s_5
$\{s_4(3), s_6(4), s_7(8)\}$	$\{s_0, s_1, s_2, s_3, s_5\}$	s_5
$\{s_6(4), s_7(8)\}$	$\{s_0, s_1, s_2, s_3, s_5, s_4\}$	s_4
$\{s_6(4), s_7(5)\}$	$\{s_0, s_1, s_2, s_3, s_5, s_4\}$	s_4
$\{s_7(5)\}$	$\{s_0, s_1, s_2, s_3, s_5, s_4, s_6\}$	s_6

Fin car s_6 est un état but !

A remarquer : l'impact sur les coûts des sommets déjà dans EnAttente (par exemple lorsqu'on traite s_2 il y a un impact sur le coût de s_4 ; idem pour s_7 lorsqu'on traite s_4).

2. Algorithme A*. Ici on va prendre en compte à la fois le coût des arcs et les heuristiques. Le choix se fera sur la fonction $f = g + h$. L'évolution des listes EnAttente et Vus au cours de l'algo est la suivante (entre parenthèses est donnée la valeur f pour chaque état e = somme des arcs depuis s_0 jusqu'à e + $h(e)$) :

EnAttente	Vus	e
$\{s_0(4)\}$	$\{\}$	
$\{\}$	$\{s_0(4)\}$	s_0
$\{s_2(4), s_1(5), s_3(6)\}$	$\{s_0(4)\}$	s_0
$\{s_1(5), s_3(6)\}$	$\{s_0(4), s_2(4)\}$	s_2
$\{s_1(5), s_3(6), s_4(8)\}$	$\{s_0(4), s_2(4)\}$	s_2
$\{s_3(6), s_4(8)\}$	$\{s_0(4), s_2(4), s_1(5)\}$	s_1
$\{s_5(5), s_3(6), s_4(8)\}$	$\{s_0(4), s_2(4), s_1(5)\}$	s_1
$\{s_3(6), s_4(8)\}$	$\{s_0(4), s_2(4), s_1(5), s_5(5)\}$	s_5
$\{s_3(6), s_4(8), s_7(8)\}$	$\{s_0(4), s_2(4), s_1(5), s_5(5)\}$	s_5
$\{s_4(8), s_7(8)\}$	$\{s_0(4), s_2(4), s_1(5), s_5(5), s_3(6)\}$	s_3
$\{s_6(4), s_4(8), s_7(8)\}$	$\{s_0(4), s_2(4), s_1(5), s_5(5), s_3(6)\}$	s_3
$\{s_4(8), s_7(8)\}$	$\{s_0(4), s_2(4), s_1(5), s_5(5), s_3(6), s_6(4)\}$	s_6

Fin car s_6 est un état but !

Concernant h, on peut dire que :
— h est coïncidente (puisque $h(s_6) = h(s_7) = 0$).
— h n'est pas monotone ; il suffit de regarder le cas de s_4 : $h(s_4) = 5$, s_7 est le fils de s_4 et on a $h(s_7) + \text{coût}(s_4, s_7) = 0 + 2$; on n'a donc pas $h(s_4) \leq h(s_7) + \text{coût}(s_4, s_7)$.
— h n'est pas minorante ; il suffit là-aussi de regarder le cas de s_4 : $h(s_4) = 5$ et $h^*(s_4) = 2$; on n'a donc pas $h(s_4) \leq h^*(s_4)$.

Avec l'algorithme A*, on a donc trouvé la meilleure solution mais en utilisant une heuristique qui n'est pas minorante (et donc rien ne nous garantissait de trouver la bonne solution !).

Solution de l'énoncé 24 page 40 : Algorithme A* avec h_1. L'évolution des listes EnAttente et Vus au cours de l'algo est la suivante (entre parenthèses est donnée la valeur f pour chaque état e = somme des arcs depuis s_0 jusqu'à $e + h_1(e)$) :

EnAttente	Vus	e
$\{s_0(0)\}$	$\{\}$	
$\{\}$	$\{s_0(0)\}$	s_0
$\{s_3(5), s_1(8), s_2(9)\}$	$\{s_0(0)\}$	s_0
$\{s_1(8), s_2(9)\}$	$\{s_0(0), s_3(5)\}$	s_3
$\{s_5(7), s_1(8), s_2(9)\}$	$\{s_0(0), s_3(5)\}$	s_3
$\{s_1(8), s_2(9)\}$	$\{s_0(0), s_3(5), s_5(7)\}$	s_5
$\{s_1(8), s_7(8), s_2(9), s4(9)\}$	$\{s_0(0), s_3(5), s_5(7)\}$	s_5
$\{s_7(8), s_2(9), s_4(9)\}$	$\{s_0(0), s_3(5), s_5(7), s_1(8)\}$	s_1
$\{s_4(4), s_7(8), s_2(9)\}$	$\{s_0(0), s_3(5), s_5(7), s_1(8)\}$	s_1
$\{s_7(8), s_2(9)\}$	$\{s_0(0), s_3(5), s_5(7), s_1(8), s_4(4)\}$	s_4
$\{s_6(4), s_5(6), s_7(8), s_2(9)\}$	$\{s_0(0), s_3(5), s_1(8), s_4(4)\}$	s_4
$\{s_5(6), s_7(8), s_2(9)\}$	$\{s_0(0), s_3(5), s_1(8), s_4(4), s_6(4)\}$	s_6

Fin car s_6 est un état but !

(6 états développés et 8 générés)

A noter : l'impact de la prise en compte de s_4 qui remet dans EnAttente le sommet s_5 qui était dans Vus (parce qu'en passant par s_4, on vient d'améliorer le coût de s_5).

Algorithme A* avec h_2. L'évolution des listes EnAttente et Vus au cours de l'algo est la suivante (entre parenthèses est donnée la valeur f pour chaque état e = somme des arcs depuis s_0 jusqu'à $e + h_2(e)$) :

EnAttente	Vus	e
$\{s_0(0)\}$	$\{\}$	
$\{\}$	$\{s_0(0)\}$	s_0
$\{s_1(5), s_3(6), s_2(9)\}$	$\{s_0(0)\}$	s_0
$\{s_3(6), s_2(9)\}$	$\{s_0(0), s_1(5)\}$	s_1
$\{s_4(4), s_3(6), s_2(9)\}$	$\{s_0(0), s_1(5)\}$	s_1
$\{s_3(6), s_2(9)\}$	$\{s_0(0), s_1(5), s_4(4)\}$	s_4
$\{s_6(4), s_5(6), s_3(6), s_2(9)\}$	$\{s_0(0), s_1(5), s_4(4)\}$	s_4
$\{s_5(6), s_3(6), s_2(9)\}$	$\{s_0(0), s_1(5), s_4(4), s_6(4)\}$	s_6

Fin car s_6 est un état but !

(4 états développés et 6 générés)

Concernant h_1 et h_2, on peut dire que, ni l'une, ni l'autre, ne sont minorantes : il suffit de regarder le cas de s_2 : $h_i(s_2) = 8$ et $h_i^*(s_2) = 6$; on n'a donc pas $h_i(s_2) \leq h_i^*(s_2)$ (pour $i = 1, 2$).

Avec l'algorithme A*, on a donc trouvé à chaque fois la meilleure solution mais en utilisant des heuristiques qui ne sont pas minorantes (et donc rien ne nous garantissait de trouver la bonne solution !).

Solution de l'énoncé 25 page 41 :

En appliquant l'algo de coût uniforme, on obtient $a - b - d$ comme chemin le plus court entre a et d (longueur 5).

Une heuristique pour appliquer le A* serait de considérer les villes dans un plan 2D avec leurs coordonnées et d'utiliser comme fonction h la distance à vol d'oiseau entre la ville courante et la ville but. On est sûr ainsi de minimiser le coût du chemin optimal.

Solution de l'énoncé 26 page 41 :

1. Ici, un état sera une séquence de villes. Soit $e = (v_1, v_2, \ldots, v_n)$, le coût de e sera la somme des distances de v_1 à v_1 en passant par toutes les villes de la séquence. Par exemple, $e = (a, b, d)$ a pour coût 12. Par convention si $e = ()$ ou $e = (v)$, on aura coût$(e) = 0$.

   L'état initial sera la séquence vide.

   Un état but sera une séquence dont la longueur = le nb total de villes (ici 4).

   Il existe un seul opérateur `ajouterVille(ville)` qui consiste à ajouter une ville à un état donné : soit $e = (v_1, v_2, \ldots, v_n)$, l'application de `ajouterVille(`$v$`)` produit un nouvel état $e' = (v_1, v_2, \ldots, v_n, v)$ dont le coût est défini par :

   $$\text{coût}(e') = \text{coût}(e) - \text{coût}(v_n, v_1) + \text{coût}(v_n, v) + \text{coût}(v, v_1).$$

2. La taille de l'espace d'état correspond au nombre de séquences de villes possibles, sachant qu'une séquence est une liste ordonnée et que l'on a 4 villes possibles. On a ainsi les séquences de longueur 0, celles de longueur 1, ..., et enfin celles de longueur 4.

   En mathématiques, le nombre de listes ordonnées de longueur k construites à partir de n éléments correspond à la notion d'arrangement : $A_n^k = \frac{n!}{(n-k)!}$.

   La taille de l'espace d'état est donc : $A_4^0 + A_4^1 + A_4^2 + A_4^3 + A_4^4$.

   Donc $\frac{4!}{(4-0)!} + \frac{4!}{(4-1)!} + \frac{4!}{(4-2)!} + \frac{4!}{(4-3)!} + \frac{4!}{(4-4)!}$.

   Il y a donc $1 + 4 + 12 + 24 + 24 = 65$ états possibles.

   On peut réduire cette taille en ne prenant pas en compte toutes les séquences : en effet, la séquence "abc" est identique à "bca" et à "cab". On a la propriété suivante : soit une séquence de longueur k, il existe k variantes de cette séquence

obtenues par décalage circulaire. Donc si on veut en garder 1 seule, il suffit de diviser le nb de séquences de longueur k par k. Cela veut dire, que dans notre cas, on peut se ramener à : $1 + 4/1 + 12/2 + 24/3 + 24/4 = 25$ états.

Et enfin, on peut aussi dans le cas précis de notre exemple, tenir compte du fait que le graphe est non orienté et donc que "abc" est identique à "cba" en terme de coût. Cela veut dire que, dès que la longueur de la séquence est strictement $>$ à 2, on peut supprimer la moitié des séquences de cette longueur (on ne le fait pas pour des séquences inférieures à 2 car cela ferait doublon avec la prise en compte des décalages circulaires. On arrive ainsi à $1 + 4 + 6 + 8/2 + 6/2 = 18$ états.

3. la méthode du coût uniforme permet de générer 16 états et d'en développer 11 (dans l'ordre : (), (a), (b), (c), (d), (ab), (bd), (ac), (cdb), $(cdba)$).

   L'état but atteint est $(cdba)$ de coût 11.

4. On constate que h est minorante. L'application du A* donnera donc la meilleure solution. Le déroulement de l'algorithme est identique à celui du coût uniforme. On trouve le même résultat en développant et générant le même nb d'états. L'heuristique, bien que minorante, ne nous a pas fait gagné de temps par rapport à la simple prise en compte du coût.

Corrigés des exercices de la section 8.4 page 47

Solution de l'énoncé 27 page 47 : Pour la fonction `nouvelle_solution`, on a l'algo 9.

Algorithme 9 : Algorithme pour `nouvelle_solution`

Données :
LesObjets : ensemble de tous les objets possibles,
Max : la valeur maximale autorisée pour le sac,
`tirageAléatoire`(bi, bs) : tirage aléatoire d'un entier dans l'intervalle [bi,bs]
Variables :
E : état résultat,
o : objet courant,
alea : variable aléatoire,
n_o : numéro de l'objet courant dans l'ensemble LesObjets
début
  E $\leftarrow \varnothing$
  alea $\leftarrow$ `tirageAléatoire`(1, |LesObjets|)
  pour *i=1 à alea* **faire**
    $n_o \leftarrow$ `tirageAléatoire`(0, |LesObjets| -1)
    $o \leftarrow$ LesObjets[n_o]
    si *p(E) + p(o)* $\leq$ *Max* **alors**
      E $\leftarrow$ E $\cup \{o\}$
    LesObjets $\leftarrow$ LesObjets $-\{o\}$
  retourner E

La notation $|X|$ correspond au cardinal de l'ensemble X (nb d'éléments ds X).
La notation $X[i]$ correspond à l'accès à l'élément en position i dans X.

Pour la fonction `eval`, on a l'algorithme 10.

Algorithme 10 : Algorithme pour `eval`

Données :
E : un état,
$v(o)$: valeur d'un objet o
Variables :
val : valeur courante,
o : objet courant
début
  val $\leftarrow 0$
  pour *chaque objet o de E* **faire**
    val $\leftarrow$ val + $v(o)$
  retourner val

Pour la fonction `voisinage`, on a l'algorithme 11 page ci-contre.

Solution de l'énoncé 28 page 47 : Dans les deux cas, c'est un problème d'optimisation avec maximisation.

Algorithme 11 : Algorithme pour `voisinage`

Données :

E : un état,
tas : ensemble de tous les objets qui ne sont pas ds E,
Max : la valeur maximale autorisée pour le sac,
`tirageAléatoire`(bi, bs) : tirage aléatoire d'un entier dans l'intervalle [bi,bs]

Variables :

EV : voisinage courant (ensemble de voisins),
E' : voisin courant,
o, o' : objets,
n_o : numéro d'objet

début

```
EV ← ∅
// génération de voisins par échange d'objet
pour chaque objet o de E faire
    n_o ← tirageAléatoire(0, |tas| −1)
    o' ← tas[n_o]
    E' = E −{o} ∪{o'}
    si p(E') ≤ Max alors
        EV ← EV ∪{E'}
// génération de voisins par ajout d'objet
pour chaque objet o de tas faire
    E' = E ∪{o}
    si p(E') ≤ Max alors
        EV ← EV ∪{E'}
// génération de voisins par suppression d'objet
pour chaque objet o de E faire
    E' = E −{o}
    EV ← EV ∪{E'}
retourner EV
```

1. Avec le SHC :
   — on part de s_0, la fonction `choisir_voisin` renvoie s_1, s_1 meilleur que s_0,
   — on repart donc de s_1, la fonction `choisir_voisin` renvoie s_2, s_2 meilleur que s_1,
   — on repart donc de s_2, la fonction `choisir_voisin` renvoie s_5, s_5 pas meilleur que s_2,
   — on repart donc de s_2, et on va y rester quel que soit le nombre de mouvements.
   En conclusion, la meilleure solution atteinte avec le SHC sera s_2.
2. avec le Tabou : liste Tabou vide au départ
   — on part de s_0, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_1, s_1 meilleur que s_0, la meilleure solution est donc s_1
   — on repart donc de s_1, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_2, s_2 meilleur que s_1, la meilleure solution est donc s_2
   — on repart donc de s_2, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_5, s_5 pas meilleur que s_2, la meilleure solution reste donc s_2 et la liste Tabou est égale à $\{s_2\}$

— mais on repart avec s_5, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_6, s_6 meilleur que s_5, la meilleure solution est donc s_6
— on repart donc de s_6, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_5, s_5 pas meilleur que s_6, la meilleure solution reste donc s_6 et la liste Tabou est égale à $\{s_2, s_6\}$
— mais on repart avec s_5, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_7, s_7 pas meilleur que s_5, la meilleure solution reste donc s_6 et la liste Tabou est égale à $\{s_2, s_6, s_5\}$
— mais on repart avec s_7, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_4, s_4 meilleur que s_7, mais la meilleure solution reste s_6
— on repart donc de s_4, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_3, s_3 meilleur que s_4, mais la meilleure solution reste s_6
— on repart donc de s_3, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_1, s_1 meilleur que s_3, mais la meilleure solution reste s_6
— on repart donc de s_1, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_3, s_3 pas meilleur que s_1, la meilleure solution reste donc s_6 et la liste Tabou est égale à $\{s_2, s_6, s_5, s_1\}$
— on repart donc de s_3, le meilleur voisin non tabou fourni par la fonction `voisinage` est s_4, s_4 pas meilleur que s_3, la meilleure solution reste donc s_6 et la liste Tabou est égale à $\{s_2, s_6, s_5, s_1, s_3\}$
— on a atteint la limite de 50% de solutions dans la liste Tabou (5 solutions sur les 9) et on s'arrête.

En conclusion, la meilleure solution atteinte avec le Tabou sera s_6.

Solution de l'énoncé 29 page 48 : Un exemple d'exécution est donné par le fichier log suivant. Dans les fichiers log n'apparaissent que les meilleures valeurs trouvées à chaque étape. Dans le cas de notre exemple, au bout de 3 essais, on a atteint un optimim local dont on ne bouge plus :

```
======== STEEPEST HILL CLIMBING :

etat courant = {} w: 0.0 $: 0.0
 voisin ameliorant = { 0} w: 4.0 $: 7.0
 voisin ameliorant = { 2} w: 8.0 $: 32.0
etat courant = { 2} w: 8.0 $: 32.0
 voisin ameliorant = { 0 2} w: 12.0 $: 39.0
 voisin ameliorant = { 2 3} w: 12.0 $: 41.0
etat courant = { 2 3} w: 12.0 $: 41.0
 voisin ameliorant = { 0 2 3} w: 16.0 $: 48.0
 voisin ameliorant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
=======(nb d'etats explores = 4)========

======== STEEPEST HILL CLIMBING avec 10 essais :

etat courant = {} w: 0.0 $: 0.0
 voisin ameliorant = { 0} w: 4.0 $: 7.0
 voisin ameliorant = { 2} w: 8.0 $: 32.0
etat courant = { 2} w: 8.0 $: 32.0
```

```
 voisin ameliorant = { 0 2} w: 12.0 $: 39.0
 voisin ameliorant = { 2 3} w: 12.0 $: 41.0
etat courant = { 2 3} w: 12.0 $: 41.0
 voisin ameliorant = { 0 2 3} w: 16.0 $: 48.0
 voisin ameliorant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
=======(nb d'etats explores = 10)========

======== STEEPEST HILL CLIMBING avec 100 essais :

etat courant = {} w: 0.0 $: 0.0
 voisin ameliorant = { 0} w: 4.0 $: 7.0
 voisin ameliorant = { 2} w: 8.0 $: 32.0
etat courant = { 2} w: 8.0 $: 32.0
 voisin ameliorant = { 0 2} w: 12.0 $: 39.0
 voisin ameliorant = { 2 3} w: 12.0 $: 41.0
etat courant = { 2 3} w: 12.0 $: 41.0
 voisin ameliorant = { 0 2 3} w: 16.0 $: 48.0
 voisin ameliorant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
...
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
etat courant = { 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
=======(nb d'etats explores = 100)========
```

On voit donc qu'on a atteint très vite un optimum local et ensuite qu'on y est resté parce qu'on ne prend en compte que le meilleur voisin à chaque étape. Cela n'est pas forcément le cas avec le Hill-Climbing qui choisit le voisin à tester de manière aléatoire (et donc ne prend pas forcément le meilleur à chaque fois). Du coup, on "monte moins vite" et on explore plus largement. Sur de petits exemples et avec un nombre d'essais suffisant, on peut même arriver à trouver la meilleure solution. Dans le fichier log suivant, on voit qu'il a fallu 12 essais pour atteindre un optimum qu'on n'arrive plus à améliorer (il s'agit ici de l'optimum global mais ce n'est pas toujours le cas).

```
======== HILL CLIMBING :
```

```
etat courant = {} w: 0.0 $: 0.0
 voisin choisi = { 5} w: 9.0 $: 8.0
etat courant = { 5} w: 9.0 $: 8.0
 voisin choisi = { 0 5} w: 13.0 $: 15.0
etat courant = { 0 5} w: 13.0 $: 15.0
 voisin choisi = { 0 1 5} w: 16.0 $: 17.0
etat courant = { 0 1 5} w: 16.0 $: 17.0
 voisin choisi = { 0 1 3 5} w: 20.0 $: 26.0
etat courant = { 0 1 3 5} w: 20.0 $: 26.0
 voisin choisi = { 0 3 5} w: 17.0 $: 24.0
{ 0 1 3 5} w: 20.0 $: 26.0
{ 0 1 3 5} w: 20.0 $: 26.0
=======(nb d'etats explores = 5)========

======== HILL CLIMBING avec 10 essais :

etat courant = {} w: 0.0 $: 0.0
 voisin choisi = { 2} w: 8.0 $: 32.0
etat courant = { 2} w: 8.0 $: 32.0
 voisin choisi = { 1 2} w: 11.0 $: 34.0
etat courant = { 1 2} w: 11.0 $: 34.0
 voisin choisi = { 1 2 5} w: 20.0 $: 42.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 1 2 4} w: 15.0 $: 36.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 2 5} w: 17.0 $: 40.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 2 4 5} w: 21.0 $: 42.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 1 5} w: 12.0 $: 10.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 1 2 4} w: 15.0 $: 36.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 1 2} w: 11.0 $: 34.0
etat courant = { 1 2 5} w: 20.0 $: 42.0
 voisin choisi = { 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
{ 2 3 5} w: 21.0 $: 49.0
=======(nb d'etats explores = 10)========

======== HILL CLIMBING avec 100 essais :

etat courant = {} w: 0.0 $: 0.0
 voisin choisi = { 4} w: 4.0 $: 2.0
etat courant = { 4} w: 4.0 $: 2.0
 voisin choisi = {} w: 0.0 $: 0.0
etat courant = { 4} w: 4.0 $: 2.0
 voisin choisi = {} w: 0.0 $: 0.0
etat courant = { 4} w: 4.0 $: 2.0
 voisin choisi = { 4 5} w: 13.0 $: 10.0
```

```
etat courant = { 4 5} w: 13.0 $: 10.0
 voisin choisi = { 0 4 5} w: 17.0 $: 17.0
etat courant = { 0 4 5} w: 17.0 $: 17.0
 voisin choisi = { 0 4} w: 8.0 $: 9.0
etat courant = { 0 4 5} w: 17.0 $: 17.0
 voisin choisi = { 0 4} w: 8.0 $: 9.0
etat courant = { 0 4 5} w: 17.0 $: 17.0
 voisin choisi = { 0 3 4 5} w: 21.0 $: 26.0
etat courant = { 0 3 4 5} w: 21.0 $: 26.0
 voisin choisi = { 1 3 4 5} w: 20.0 $: 21.0
etat courant = { 0 3 4 5} w: 21.0 $: 26.0
 voisin choisi = { 0 2 3 4} w: 20.0 $: 50.0
etat courant = { 0 2 3 4} w: 20.0 $: 50.0
 voisin choisi = { 2 3 4} w: 16.0 $: 43.0
etat courant = { 0 2 3 4} w: 20.0 $: 50.0
 voisin choisi = { 0 1 2 3 4} w: 23.0 $: 52.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 3 4} w: 15.0 $: 20.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 3} w: 19.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 2 3 4} w: 20.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 3} w: 19.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 4} w: 19.0 $: 43.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 2 3 4} w: 20.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 4} w: 19.0 $: 43.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 1 2 3 4} w: 19.0 $: 45.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 1 2 3 4} w: 19.0 $: 45.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 3} w: 19.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 2 3 4} w: 20.0 $: 50.0
...
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 3 4} w: 15.0 $: 20.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 2 3} w: 19.0 $: 50.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 3 4} w: 15.0 $: 20.0
etat courant = { 0 1 2 3 4} w: 23.0 $: 52.0
 voisin choisi = { 0 1 3 4} w: 15.0 $: 20.0
{ 0 1 2 3 4} w: 23.0 $: 52.0
{ 0 1 2 3 4} w: 23.0 $: 52.0
=======(nb d'etats explores = 100)========
```

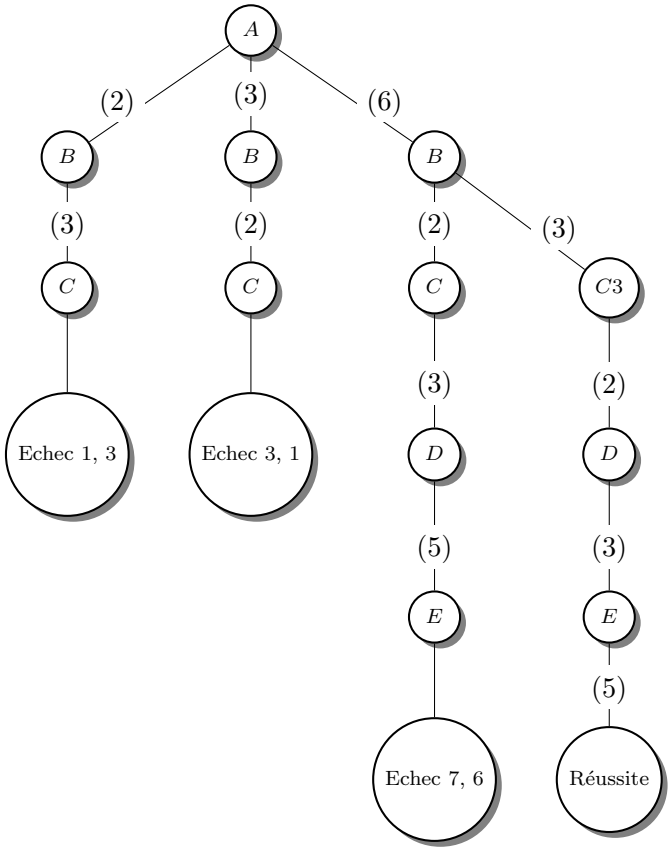
Corrigés des exercices de la section 9.4 page 57

Solution de l'énoncé 30 page 57 : On peut choisir 1 variable par composant : p, m et d.

Les domaines sont les suivants :
— $D(p) = \{p_1, p_2, p_3\}$,
— $D(m) = \{m_1, m_2, m_3, m_4\}$,
— $D(d) = \{d_1, d_2, d_3\}$.
Et les contraintes sont les suivantes (tuples interdits) :
— $C_{pd} = \{(p_1, d_3)\}$,
— $C_{pm} = \{(p_2, m_1), (p_3, m_1), (p_3, m_2)\}$,
— $C_{pmd} = \{(p_2, m_2, d_1), (p_2, m_2, d_3)\}$.

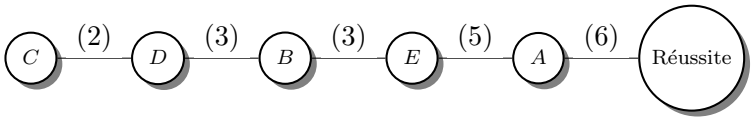
Solution de l'énoncé 31 page 57 :

1. Appliquez le backtrack en prenant les variables par ordre alphabétique et les valeurs par ordre croissant. Vous donnerez uniquement l'arbre d'affectation des variables en précisant les impasses (avec le numéro de la contrainte violée) et en arrêtant à la première affectation valide.



2. Faire la même chose avec un ordonnancement préalable : ordre croissant sur la taille des domaines, puis si égalité ordre décroissant sur le nb de contraintes et enfin ordre alphabétique.

   Cet ordonnancement produit l'ordre C-D-B-E-A et le déroulement du BT produit :



3. Pour chaque variable donnez la valeur la moins contraignante en expliquant.

   Pour A, c'est 6 (1 seul voisin concerné alors que toutes les autres valeurs concernent au moins 2 voisins)

   Pour B, toutes pareilles

   Pour C, c'est 2 (2 voisins concernés alors que 3 concerne les 3 voisins)

   Pour D, c'est 5 (1 voisin concerné alors que 3 concerne 2 voisins)

   Pour E, toutes pareilles

4. Supposons qu'on assigne la valeur 2 à B. Que donnerait une propagation de contraintes jusqu'à ce qu'il n'y ait plus de changement ?

   Par propagation, le domaine de A passe à $\{3, 6\}$ et celui de C à $\{3\}$.

   Donc C est affectée avec 3 et par propagation, le domaine de A passe à $\{6\}$ et celui de D à $\{5\}$.

   Donc A est affectée avec 6 et par propagation, le domaine de E passe à $\{5\}$.

   Donc D est affectée avec 5 et par propagation, le domaine de E passe à $\varnothing$. Donc ECHEC.

Solution de l'énoncé 32 page 58 : Pour modéliser ce pb sous la forme d'un CSP, il faut répondre aux questions suivantes :
— Quelles variables ?
— Quel domaine pour quelle variable ?
— Quelles contraintes ?
On pourrait envisager a priori 6 variables avec les domaines suivants :
— maison $\in \{1, 2, 3, 4, 5\}$ avec un ordre précis pour prendre en compte la signification des expressions "après", "à côté de", ...
— couleur $\in$ {bleu, vert, rouge, blanc, jaune},
— métier $\in$ {diplomate, docteur, peintre, sculpteur, violoniste},
— nationalité $\in$ {anglais, italien, japonais, norvégien, espagnol},
— boisson $\in$ {café, jus, lait, thé, eau},
— animal $\in$ {chien, renard, cheval, escargot, zèbre}.
Par contre, il va du coup être très difficile de résoudre le pb dans sa globalité. On pourra juste avoir la description concernant une maison et son contenu, mais pas celle de toutes les maisons (à part si on cherche toutes les affectations possibles).

D'autre part, il va être aussi très difficile d'établir le lien entre le numéro de la maison et le reste. Et pourtant, la couleur, le métier, ... sont des propriétés liées au numéro de la maison. Ce qui nous intéresse est de dire par exemple où se trouve le diplomate : dans la maison 1, 2, 3, 4, ou 5 ?

La solution correcte va donc consister à considérer non plus 6 variables avec des domaines tous différents, mais 25 variables (bleu, vert, ..., diplomate, ..., anglais, ..., café, ..., chien, ...) ayant toutes le même domaine $\{1, 2, 3, 4, 5\}$ et pouvant être vues comme partitionnées en 5 familles de variables.

Du coup, en ce qui concerne les contraintes, on a les éléments suivants :
— Tout d'abord une contrainte générale : chien $\neq$ chat (la valeur prise par la variable chien doit être différente de celle prise par la variable chat), chat $\neq$ cheval, etc sur les 5 familles de variables.

— Puis ensuite la prise en compte des informations fournies, ce qui va permettre d'établir un lien entre les valeurs de certaines variables :
  — L'anglais habite dans la maison rouge.
    rouge=anglais
  — L'espagnol a un chien.
    espagnol=chien
  — Le japonais est peintre.
    japonais=peintre
  — L'italien boit du thé.
    italien=thé
  — Le norvégien habite la première maison.
    norvegien=1
  — L'habitant de la maison verte boit du café.
    vert=café
  — La maison verte est après la blanche.
    vert $\geq$ blanche+1
  — Le sculpteur élève des escargots.
    sculpteur=escargot
  — Le diplomate habite la maison jaune.
    jaune=diplomate
  — On boit du lait dans la 3e maison.
    lait=3
  — La maison du norvégien est à côté de la bleue.
    |norvegien-bleue| =1
  — Le violoniste boit du jus de fruit.
    violoniste=jus
  — Le renard est dans la maison après celle du docteur.
    renard=docteur+1
  — Le cheval est la maison après celle du diplomate.
    cheval=diplomate+1
  — Le zèbre est dans la maison blanche.
    zebre=blanche
  — Une des personnes boit de l'eau. Ici, cela ne correspond à aucune contrainte. Cette information a juste servi à ajouter l'eau à la liste des boissons.

La résolution du problème peut alors se faire par une recherche arborescente dans les valeurs possibles (algo du backtrack). Par contre, si on utilise aussi la **propagation des contraintes**, cela permet de simplifier le problème en réduisant les domaines des variables.

Par exemple, suite aux contraintes on sait que la valeur de norvégien = 1, on a donc
— 1 devient impossible pour espagnol, italien, etc
— bleu $\in \{0, 2\} \cap D(bleu)$ donc la valeur de bleu est forcément 2.
Cette réduction des domaines des variables permet de ne pas tester des valeurs dont on sait qu'elles ne sont pas comptatibles. Cette approche correspond à une amélioration de l'algorithme de backtrack qu'on appelle le *forward-checking*.

La solution finale du problème est la suivante :

animal chien = 3, renard = 5, cheval = 4, escargot = 2, zèbre = 1,

couleur bleu = 2, vert = 5, rouge = 4, blanc = 1, jaune = 3,

boisson café = 5, jus = 1, lait = 3, thé = 2, eau = 4,

nationalité anglais = 4, italien = 2, japonais = 5, norvégien = 1, espagnol = 3,
profession diplomate = 3, docteur = 4, peintre = 5, sculpteur = 2, violiniste = 1

Corrigés des exercices de la section 10.4 page 64

Solution de l'énoncé 33 page 64 : Soit un graphe $G = (X, E)$. On associe à chaque sommet x de G un vecteur binaire à K dimensions $v_x = (v_x^1, \ldots, v_x^K)$, où K est une borne max du nombre chromatique (au max $K = |X|$).

Comme on veut minimiser le nombre de couleurs, on ajoute une variable binaire w_l par couleur $l = 1, \ldots, K$ indiquant si cette couleur a été utilisée ou pas.

Le problème est alors équivalent à :

$$\text{Minimiser } \sum_{l=1}^{K} w_l$$
avec les contraintes :
$$\forall x \in X,\ \sum_{i=1}^{K} v_x^i = 1$$
$$\forall (x, y) \in E \text{ et } 1 \leq l \leq K,\ v_x^l + v_y^l \leq w_l$$
et avec $\forall x \in X$ et $1 \leq l \leq K$, $v_x^l \in \{0, 1\}$ et $w_l \in \{0, 1\}$

La première contrainte oblige chaque commet à n'être affecté que par une seule couleur. Et la seconde contrainte interdit à deux sommets adjacents d'avoir la même couleur.

Remarque : cette représentation a une taille polynomiale avec dans le pire des cas $n^2 + n$ variables (n étant le nb de sommets dans G). Toutefois, elle va s'avérer en pratique très difficile à résoudre.

Solution de l'énoncé 34 page 64 : Considérons le graphe complet $G = (X, E)$ à n sommets, c'est-à-dire le graphe d'ordre n où tous les sommets sont adjacents deux à deux. On suppose de plus qu'on a défini une valuation sur l'ensemble E des arêtes de ce graphe, autrement dit une application p de E dans $\mathbb{R}$ qu'on appelle "poids", le poids d'une arête (x, y) étant noté p_{xy}.

Rappelons que le pb du TSP consiste à rechercher un cycle hamiltonien de poids minimal dans le graphe (cycle hamiltonien : cycle qui passe par chaque sommet du graphe une et une seule fois). Associons donc à chaque arête (x, y) de G une variable bivalente v_{xy} qui vaut 1 si on garde l'arête pour constituer le cycle hamiltonien, 0 sinon.

Les contraintes correspondant à la constitution d'un cycle hamiltonien sont les suivantes :

— pour tout sommet x, la somme des valeurs des variables associées aux arêtes ayant x comme extrémité est égale à deux, ceci traduisant le fait que dans un cycle hamiltonien tout sommet est de degré deux ;

— pour tout sous-ensemble Y de X autre que X, le nombre d'arêtes ayant ses deux extrémités dans Y est strictement plus petit que $|Y|$; cette condition implique que l'ensemble des arêtes gardées ne se décompose pas en plusieurs cycles.

Toutes ces contraintes s'expriment bien linéairement par rapport à l'ensemble des variables du problème, comme le montre la forme suivante :

$$\text{Minimiser } \sum_{(x,y) \in E} p_{xy} \times v_{xy}$$
avec les contraintes :
$$\forall x \in X,\ \sum_{y \in X} v_{xy} = 2$$
$$\forall Y \subset X,\ Y \neq X,\ \sum_{(x,y) \in E \cap (Y \times Y)} v_{xy} < |Y|$$
et avec $\forall (x, y) \in E$, $v_{xy} \in \{0, 1\}$