15/12/2014

Bureau d'études : Intelligence Artificielle

VEYSSEIRE Daniel
& FABRE Mickaël
M2 IRR UNIVERSITE PAUL SABATIER

Table des matières

I. Feuille de projet n°1 : SAT4J

1. Vérifier si un ensemble de faits est cohérent

On cherche à établir la compatibilité des témoignages, pour ce faire nous prenons les variables suivantes : x1 = Dupont est coupable, x2 = Legrand est coupable, x3 = Martin est coupable.

On modélise ensuite sur SAT4J notre problème afin de savoir si les témoignages sont compatibles (ils ne se contredisent pas). Ainsi si on trouve une solution à ce problème, ceci voudra dire qu'ils le sont.

Programme SAT4J :

```
p cnf 3 5
2 0
-3 0
-1 3 0
-3 0
2 1 0
```

Solution : SATISFIABLE
-1 2 -3 0

Ce qui veut dire que la solution trouvé est : Dupont est innocent, Legrand est coupable et Martin est innocent. Le problème de cette déduction via SAT4J réside dans le fait qu'il puisse y avoir de multiples solutions. Dans notre problème, nous ne pouvons donc pas être sûr que la solution trouvé est la bonne, simplement que les faits sont cohérents et possibles.

On cherche donc maintenant à déduire que Legrand est coupable et que Martin et Dupont sont innocents. Pour ce faire nous allons ajouter au problème une contrainte qui est que Legrand est innocent OU Martin coupable OU Dupont coupable (la négation de la solution précédente). Ainsi, si une solution est trouvée, cela voudra dire que dans la phrase « Legrand est coupable, Martin et Dupont sont innocents », au moins un des faits peut être faux tout en ayant toujours des témoignages compatibles. En revanche, si le problème n'a pas de solution, alors on pourra dire avec certitude que cette affirmation est exacte (a condition qu'aucun n'est menti).

Programme SAT4J :

```
p cnf 3 6
2 0
-3 0
-1 3 0
-3 0
2 1 0
-2 1 3 0
```

Solution : UNSATISFIABLE
On a donc prouvé que Legrand est coupable et que Martin et Dupont sont innocents.

2. Décider si des raisonnements sont valides

Comme expliqué précédemment SAT4J ne renvoie qu'une solution parmi différentes possibles. Or dans notre cas nous cherchons à déterminer si les discours sont vrais dans tous les

cas possible, aussi nous allons modéliser le problème contraire et s'il est insatisfiable ceci voudra dire que le discours de base est vrai dans tous les cas.

Nous prenons ici : x1 = E et x2 = R.

Programme SAT4J :

```
p cnf 2 3
-1 2 0
-2 0
1 0
```

Solution : UNSATISFIABLE

Nous avons donc prouvé que le premier discours est vrai dans tous les cas, puisque ça négation est fausse dans tous les cas.

Nous raisonnons ensuite de la même manière avec le problème suivant et les variables : x1 = W, x2 = S et x3 = M.

Programme SAT4J :

```
p cnf 3 7
-1 2 0
-2 1 0
-2 3 0
-1 3 0
-2 3 0
3 0
-1 0
```

Solution : -1 -2 3 0

Le problème a donc une solution, ce qui signifie que le discours n'est pas vrai dans toutes les circonstances, si on ne peut faire ni confiance à W ni à S mais à Mario la proposition n'est pas vérifiée.

3. Coloration

1. Exercice 1 graphe

On cherche maintenant à utiliser SAT4J afin de répondre à un problème de coloration et pour commencer un problème de coloration d'un graphe de 3 sommet A, B et C que nous cherchons à colorer de 2 couleurs différentes de manière à ce que 2 sommets reliés par un arc ne soient pas de la même couleur. Avec les arcs (A,B) et (A,C) et les variables: x1 = A1, x2 = A2, x3 = B1, x4 = B2, x5 = C1 et x6 = C2. (A1 représente la couleur 1 pour le sommet A et A2 le couleur 2 pour le sommet 2).

Les 6 premières lignes représentent le domaine (soit couleur 1 soit couleur 2) et les autres représentent les contraintes des arcs de notre problème.

Programme SAT4J :

```
p cnf 6 10
1 2 0
-1 -2 0
3 4 0
-3 -4 0
5 6 0
-5 -6 0

-1 -3 0
-2 -4 0

-1 -5 0
-2 -6 0
```

Solution : -1 2 3 -4 5 -6 0

Nous avons donc résolu notre problème avec A de couleur 2, B de couleur 1 et C de couleur 1.

2. Exercice 2 parseur

Nous avons écrit un parseur en Java qui, à partir du fichier .col et un entier k, créer le fichier DIMACS CNF associé. Les contraintes de domaines (et les différentes variables) sont déduites de k et du nombre de sommets, et les contraintes d'arcs sont déduites à partir des arcs renseigné. Pour le fichier "grapheABAC.col'', pour k=1 le problème est insatisfiable, pour k=2 on retrouve le fichier codé en 1 donc la même solution.

3. Exercice 3 graphe

a) Le fichier col correspondant au problème 3.a est le suivant:

```
p edge 7 12
e 1 2
e 1 3
e 1 4
e 2 3
e 2 4
e 2 5
e 3 4
e 3 5
e 3 6
e 4 6
e 5 7
e 6 7
```

Le parseur a créé le fichier CNF et les solutions sont insatisfiables jusqu'à k=4.

La solution trouvée est :

-1 2 -3 -4 -5 -6 -7 8 -9 -10 11 -12 13 -14 -15 -16 -17 18 -19 -20 -21 22 -23 -24 25 -26 -27 -28

Ce qui signifie : S1 de couleur 2, S2 couleur 4, S3 couleur 3, S4 couleur 1, S5 couleur 2, S6 de couleur 2, S7 de couleur 1.

Pour le problème 3.b, on modélise le problème ainsi :

```
p edge 8 19
e 1 4
e 1 7
e 2 1
e 2 5
e 2 6
e 2 7
e 3 1
e 3 4
e 3 6
e 4 1
e 4 5
e 4 8
e 5 2
e 5 4
e 6 5
e 7 3
e 7 5
e 8 1
e 8 3
```

Le nombre minimum d'aquarium est de quatre. Une solution trouvé par SAT4J est de mettre les poissons F,G,H dans l'aquarium 1 ; C et E dans l 'aquarium 2; B et D dans l'aquarium 3 ; A dans l'aquarium 4.

4. Exercice 4 sudoku

Un sudoku 4x4 peut être considéré comme un graphe à 16 sommets. Les sommets situés sur la même ligne, la même zone ou la même colonne ont des arcs entre eux. Chaque sommet est relié à 7 autres sommets, il y a donc (7*16)/2 = 56. On divise par deux car les arcs sont non orientés.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

On écrit le fichier .col correspondant, puis on utilise le parseur pour avoir le fichier CNF. Il faut maintenant ajouter les cases déjà remplis. C'est à dire rajouter les quatre lignes suivantes à la fin du fichier CNF.

```
4 0
17 0
48 0
63 0
```

Puis changer éventuellement l'entête

```
P cnf 64 336
```

par

```
p cnf 64 340
```

Même si SAT4J ne semble pas poser problème si on n'y pense pas. La solution trouvée est la suivante :

4	2	3	1
1	3	4	2
3	1	2	4
2	4	1	3

5. Exemples

a. Solution trouvé pour k=4, le cnf contient 44 sommets et 157 contraintes, résolu en 0.011 secondes.

b. Solution trouvé pour k=3, le cnf contient 60 sommets et 278 contraintes, résolu en 0.013 secondes.

c. Solution trouvé pour k=5, le cnf contient 2250 sommets et 33520 contraintes, résolu en 0.315 secondes.

SAT4J semble donc fournir d'excellentes performances.

4. Utilisation de SAT4J-CSP

On cherche à résoudre sous formes de CSP le problème du fermier aux 7 animaux. On utilise donc le programme XML_CSP suivant afin de modéliser notre problème.
Domaine : 7 Animaux.
Variables : Poules et Cochons.
Contraintes : 22 pattes.

Fichier XML_CSP :

```
<?xml version="1.0"?>

-<instance>
<presentation format="XCSP 2.0" name="PouleCochon"> Put here the
description of the instance </presentation>

-<domains nbDomains="1">
<domain name="nb_animal" nbValues="8"> 0..7 </domain>
</domains>

-<variables nbVariables="2">
<variable name="poule" domain="nb_animal"/>
<variable name="cochon" domain="nb_animal"/>
</variables>

-<predicates nbPredicates="1">
-<predicate name="p1">
<parameters> int a int x int b int y int z </parameters>
-<expression>
<functional> eq( add(mul(a,x),mul(b,y)) , z) </functional>
</expression>
</predicate>
</predicates>

-<constraints nbConstraints="2">
-<constraint name="septanimaux" reference="p1" scope="poule cochon"
arity="2">
<parameters> poule 1 cochon 1 7 </parameters>
</constraint>
-<constraint name="vingtdeuxpattes" reference="p1" scope="poule cochon"
arity="2">
<parameters> poule 2 cochon 4 22 </parameters>
</constraint>
</constraints>
</instance>
```

On trouve pour solution qu'il y a 4 cochons et 3 Poules.

Nous cherchons ensuite à résoudre le problème du shopping.
Domaine : Prix de 0 à 8.
Variables : Objets 1 à 4
Contraintes : prix de la somme des objets et prix de la multiplication des objets égal à 8 €.

Fichier XML_CSP :

```
<?xml version="1.0"?>

-<instance>
<presentation format="XCSP 2.0" name="Shopping"> </presentation>
```

```
-<domains nbDomains="1">
<domain name="prix" nbValues="9"> 0..8 </domain>
</domains>

-<variables nbVariables="4">
<variable name="objetun" domain="prix"/>
<variable name="objetdeux" domain="prix"/>
<variable name="objettrois" domain="prix"/>
<variable name="objetquatre" domain="prix"/>
</variables>

-<predicates nbPredicates="3">
-<predicate name="p1">
<parameters> int a int x int b int y int z </parameters>
-<expression>
<functional> eq( add(mul(a,x),mul(b,y)) , z) </functional>
</expression>
</predicate>
-<predicate name="p2">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(add(add(a,b),add(c,d)),e) </functional>
</expression>
</predicate>
-<predicate name="p3">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(mul(mul(a,b),mul(c,d)),e) </functional>
</expression>
</predicate>
</predicates>

-<constraints nbConstraints="2">
-<constraint name="addition" reference="p2" scope="objetun objetdeux
objettrois objetquatre" arity="5">
<parameters> objetun objetdeux objettrois objetquatre 8 </parameters>
</constraint>
-<constraint name="multiplication" reference="p3" scope="objetun
objetdeux objettrois objetquatre" arity="5">
<parameters> objetun objetdeux objettrois objetquatre 8 </parameters>
</constraint>
</constraints>
</instance>
```

La solution proposée est : objet 1 coûte 1€, objet 2 coûte 2€, objet 3 coûte 4€ et objet 5 coûte 1€.
L'idée maintenant est de résoudre le même problème avec pour coût total 7.11€ au lieu de 8€. Pour simplifier la résolution, nous multiplions toutes les valeurs par 100 de manière à se ramener à des entiers uniquement.

Fichier XML_CSP :

```
<?xml version="1.0"?>

-<instance>
<presentation format="XCSP 2.0" name="Shopping"> </presentation>

-<domains nbDomains="1">
<domain name="prix" nbValues="712"> 0..711 </domain>
```

```
</domains>

<variables nbVariables="4">
<variable name="objetun" domain="prix"/>
<variable name="objetdeux" domain="prix"/>
<variable name="objettrois" domain="prix"/>
<variable name="objetquatre" domain="prix"/>
</variables>

-<predicates nbPredicates="3">
-<predicate name="p1">
<parameters> int a int x int b int y int z </parameters>
-<expression>
<functional> eq( add(mul(a,x),mul(b,y)) , z) </functional>
</expression>
</predicate>
-<predicate name="p2">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(add(add(a,b),add(c,d)),e) </functional>
</expression>

</predicate>
-<predicate name="p3">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(mul(mul(a,b),mul(c,d)),e) </functional>
</expression>
</predicate>
</predicates>

-<constraints nbConstraints="2">
-<constraint name="addition" reference="p2" scope="objetun objetdeux
objettrois objetquatre" arity="5">
<parameters> objetun objetdeux objettrois objetquatre 711 </parameters>
</constraint>
-<constraint name="multiplication" reference="p3" scope="objetun
objetdeux objettrois objetquatre" arity="5">
<parameters> objetun objetdeux objettrois objetquatre 711 </parameters>
</constraint>
</constraints>
</instance>
```

Malheureusement dans ce cas-là le temps de calcul est trop élevé pour trouver une solution.

Le dernier problème de CSP proposé porte sur la coloration avec 3 couleurs d'un graphe à 4 sommets et 5 arcs.

Domaine : 3 couleurs.

Variables : 4 couleurs des sommets

Contraintes : 5 arcs

Fichier XML_CSP :

```
<?xml version="1.0"?>
-<instance>
<presentation format="XCSP 2.0" name="Coloration"> </presentation>

-<domains nbDomains="1">
<domain name="couleur" nbValues="3"> 0..2 </domain>
</domains>
```

```
-<variables nbVariables="4">
<variable name="a" domain="couleur"/>
<variable name="b" domain="couleur"/>
<variable name="c" domain="couleur"/>
<variable name="d" domain="couleur"/>
</variables>

-<predicates nbPredicates="4">
-<predicate name="p1">
<parameters> int a int x int b int y int z </parameters>
-<expression>
<functional> eq( add(mul(a,x),mul(b,y)) , z) </functional>
</expression>
</predicate>
-<predicate name="p2">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(add(add(a,b),add(c,d)),e) </functional>
</expression>
</predicate>
-<predicate name="p3">
<parameters> int a int b int c int d int e </parameters>
-<expression>
<functional> eq(mul(mul(a,b),mul(c,d)),e) </functional>
</expression>
</predicate>
-<predicate name="p4">
<parameters> int a int b </parameters>
-<expression>
<functional> ne(a,b) </functional>
</expression>
</predicate>
</predicates>

-<constraints nbConstraints="5">
-<constraint name="arcab" reference="p4" scope="a b" arity="2">
<parameters> a b </parameters>
</constraint>
-<constraint name="arcac" reference="p4" scope="a c" arity="2">
<parameters> a c </parameters>
</constraint>
-<constraint name="arcbc" reference="p4" scope="b c" arity="2">
<parameters> b c </parameters>
</constraint>
-<constraint name="arcbd" reference="p4" scope="b d" arity="2">
<parameters> b d </parameters>
</constraint>
-<constraint name="arccd" reference="p4" scope="c d" arity="2">
<parameters> c d </parameters>
</constraint>
</constraints>
</instance>
```

La solution proposée est A de couleur 1, B de couleur 0, C de couleur 2 et D de couleur 1.

II. Feuille de projet n°2 : Problèmes de planification

Tout au long de cette partie nous allons utiliser le langage PDDL afin de modéliser et résoudre les problèmes à l'aide du planificateur FF.

1) Le plan obtenu avec le fichier wumpus.pddl n'est pas correct car on fait bouger l'or au lieu de l'agent, alors que ce n'est pas censé être autorisé.
2) Le plan obtenu avec le fichier wumpus2.pddl n'est pas correct non plus car l'agent passe par la case du Wumpus et devrait donc être tué par ce dernier.
3) Avec la version de wumpus3.pddl le Wumpus est déplacé or nous n'avons pas le droit de contrôler ses déplacements. De plus l'agent se retrouve sur la même case que le Wumpus à une étape du plan.
4) La version wumpus4.pddl est la version correcte à utiliser. En effet l'agent se déplace correctement, tue le Wumpus, récupère l'or et revient à sa position initiale.
5)

a) Code PDDL permettant de représenter le domaine des Tours de Hanoi :

```
(define (domain hanoi)

  (:requirements :strips)
  (:predicates
   (libre ?what)
   (sur ?disque ?what)
   (pluspetit ?disque1 ?disque2) )

  (:action deplacer
   :parameters (?disque1 ?from ?what)
   :precondition (and (libre ?disque1)
                      (libre ?what)
                      (pluspetit ?disque1 ?what)
                      (sur ?disque1 ?from)
                       )
   :effect (and (not (libre ?what))
                (sur ?disque1 ?what)
                (not (sur ?disque1 ?from))
                (libre ?from)
                )  )
)
```

b)code PDDL représentant le problème avec 4 disques :

```
(define (problem hanoi-pb)
  (:domain hanoi)
  (:objects D1 D2 D3 D4
            P1 P2 P3)
  (:init (sur D4 P1)
      (sur D3 D4)
      (sur D2 D3)
      (sur D1 D2)
      (libre P2)
      (libre P3)
      (libre D1)
      (pluspetit D1 D2)
      (pluspetit D2 D3)
      (pluspetit D3 D4)
      (pluspetit D1 D3)
      (pluspetit D1 D4)
      (pluspetit D2 D4)
      (pluspetit D1 P1)
      (pluspetit D2 P1)
      (pluspetit D3 P1)
      (pluspetit D4 P1)
      (pluspetit D1 P2)
      (pluspetit D2 P2)
      (pluspetit D3 P2)
      (pluspetit D4 P2)
      (pluspetit D1 P3)
      (pluspetit D2 P3)
      (pluspetit D3 P3)
      (pluspetit D4 P3))
  (:goal (and (libre P2)
(libre P1)(libre D1)
(sur D4 P3)(sur D3 D4)
(sur D2 D3)(sur D1 D2))
)
  )
```

c) résolution du problème avec FF :

```
0: DEPLACER D1 D2 P2
1: DEPLACER D2 D3 P3
2: DEPLACER D1 P2 D2
3: DEPLACER D3 D4 P2
4: DEPLACER D1 D2 D4
5: DEPLACER D2 P3 D3
6: DEPLACER D1 D4 D2
7: DEPLACER D4 P1 P3
8: DEPLACER D1 D2 D4
9: DEPLACER D2 D3 P1
10: DEPLACER D1 D4 D2
11: DEPLACER D3 P2 D4
12: DEPLACER D1 D2 P2
13: DEPLACER D2 P1 D3
14: DEPLACER D1 P2 D2
```

6) a) Code du domaine de la satisfiabilité des formules propositionnelles à trois variables :

```
(define (domain sat3)
  (:requirements :strips)
  (:predicates
   (vrai ?what)
   (faux ?what))
  (:action vrai
    :parameters (?what)
    :effect (and (not (faux ?what))
                 (vrai ?what))
   )

  (:action faux
    :parameters (?what)
    :effect (and (not (vrai ?what))
                 (faux ?what))
   )
 )
```

b) Code PDDL correspondant à la formule suivante : *((x1 ou x2 ou x3) et (non x2) et (non x1 ou non x2) et (non x1 ou non x3) et (non x2 ou non x3)) :*

```
(define (problem sat3-pb)
  (:domain sat3)
  (:objects X1 X2 X3
        )
  (:init (vrai X1)
      (vrai X2)
      (vrai X3)
  )
  (:goal (and (or (vrai X1)
                  (vrai X2)
                  (vrai X3))
              (faux X2)
              (or(faux X1) (faux X2))
              (or(faux X1) (faux X3))
              (or(faux X2) (faux X3))
      )
) )
```

c) Résolution du problème avec FF :

```
0: FAUX X2
1: FAUX X3
2: REACH-GOAL
```

La solution trouvée est donc x1=vrai, x2 = faux et x3 = faux

7) On désire maintenant répondre au problème de CSP de coloration avec 3 couleurs de 4 sommets ayants 5 arcs.

Nous modélisons le domaine avec le code PDDL suivant :

```
(define (domain csp)
(:requirements :strips)
(:predicates
   (colorie ?sommet)
   (rouge ?sommet)
   (bleu ?sommet)
   (vert ?sommet)
   (arc ?sommet1 ?sommet2))
(:action rouge
   :parameters (?sommet)
   :precondition (not (colorie ?sommet))
   :effect (and (colorie ?sommet)
                (rouge ?sommet)))
(:action vert
    :parameters (?sommet)
    :precondition (not (colorie ?sommet))
    :effect (and (colorie ?sommet)
                 (vert ?sommet)))
(:action bleu
    :parameters (?sommet)
    :precondition (not (colorie ?sommet))
    :effect (and (colorie ?sommet)
                 (bleu ?sommet)))
(:action arc
```

La solution trouvée est :

```
0: ROUGE A
1: VERT B
2: ARC A B
3: BLEU C
4: ARC A C
5: ARC B C
6: ROUGE D
7: ARC B D
8: ARC C D
```

```
    :parameters (?sommet1 ?sommet2)
    :precondition (and
      (colorie ?sommet1)
      (colorie ?sommet2)
      (not (and (rouge ?sommet1) (rouge ?sommet2))
      (not (and (vert ?sommet1) (vert ?sommet2)) )
      (not (and (bleu ?sommet1) (bleu ?sommet2)) ))
    :effect
      (arc ?sommet1 ?sommet2)))
```

8)9)10) Le wumpus 4 gère déjà ces problème

11)

Modélisation du problème de taquin 2x3 :

```
(define (domain taquin)
  (:requirements :strips)
  (:predicates
   (libre ?square)
   (apion ?square ?pion)
   (adj ?square1 ?square2)
  )
  (:action move
   :parameters (?square_1 ?square_2 ?pion)
   :precondition (and
                  (adj ?square_1 ?square_2)
                  (libre ?square_2)
                  (apion ?square_1 ?pion))
   :effect (and (not(libre ?square_2))
                (apion ?square_2 ?pion)
                (not(apion ?square_1 ?pion)
                (libre ?square_1)
   ))

)
```

Solution trouvé :

```
0: MOVE SQ-1-2 SQ-1-1 P1
 1: MOVE SQ-1-3 SQ-1-2 P2
 2: MOVE SQ-2-3 SQ-1-3 P3
```

définition du problème

```
(define (problem taquin-pb
  (:domain taquin)
  (:objects
   sq-1-1 sq-1-2 sq-1-3
   sq-2-1 sq-2-2 sq-2-3
   p1 p2 p3 p4 p5
   )

  (:init
   (adj sq-1-1 sq-1-2)
   (adj sq-1-2 sq-1-1)
   (adj sq-1-2 sq-1-3)
   (adj sq-1-3 sq-1-2)
   (adj sq-2-1 sq-2-2)
   (adj sq-2-2 sq-2-1)
   (adj sq-2-2 sq-2-3)
   (adj sq-2-3 sq-2-2)
   (adj sq-1-1 sq-2-1)
   (adj sq-2-1 sq-1-1)
   (adj sq-1-2 sq-2-2)
   (adj sq-2-2 sq-1-2)
   (adj sq-1-3 sq-2-3)
   (adj sq-2-3 sq-1-3)
   (libre sq-1-1)
   (apion sq-1-2 p1)
   (apion sq-1-3 p2)
   (apion sq-2-1 p4)
   (apion sq-2-2 p5)
   (apion sq-2-3 p3))
  (:goal (and
    (apion sq-1-1 p1)
(apion sq-1-2 p2)
(apion sq-1-3 p3)
(apion sq-2-1 p4)
(apion sq-2-2 p5)
(libre sq-2-3)
)))
```

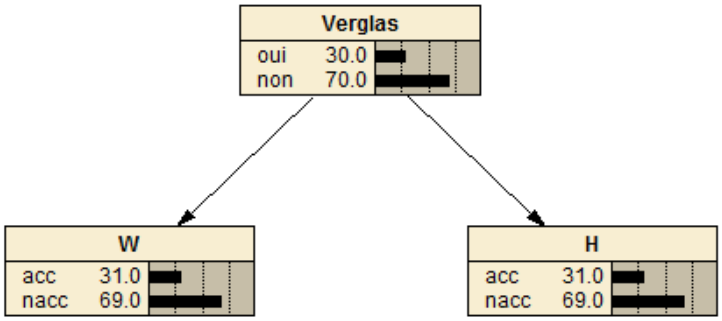
III. Feuille de projet n°3 : Réseaux bayésiens

Lors de cette partie nous allons à apprendre à utiliser le logiciel Netica nous permettant de modéliser des réseaux bayésiens et ainsi mieux appréhender les problèmes posés, en voyant leur évolution de manière dynamique.

1. Créer un réseau bayésien

William et Hector :

a) Voici le réseau bayésien correspondant au problème posé :

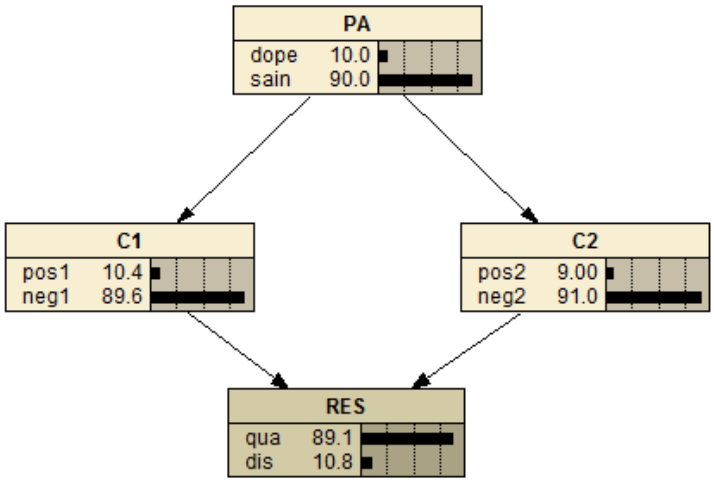


b) On apprend que William a eu un accident, la probabilité de verglas passe à 77.4% et la probabilité d'accident de Hector est devenue 64.2%.

c) On constate qu'il n'y a pas de verglas, la nouvelle probabilité d'accident pour Hector est 10%.

Contrôle antidopage :

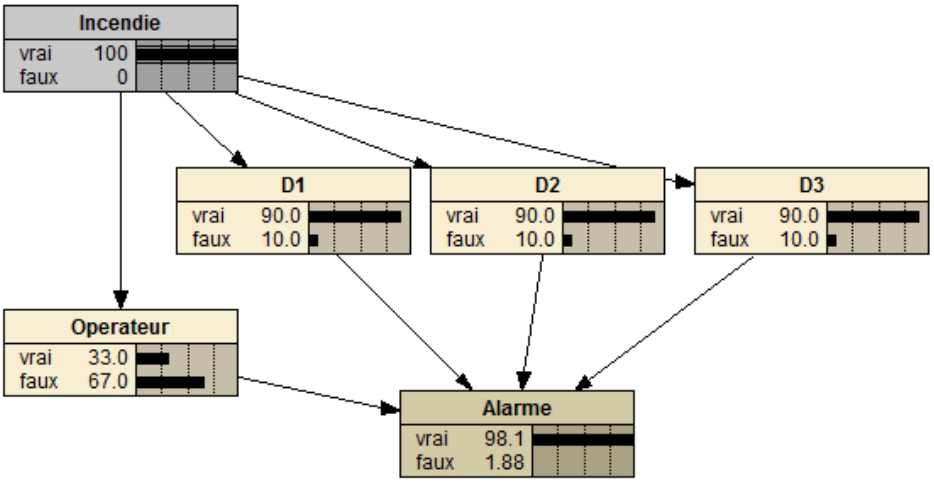
a) Réseau bayésien correspondant au problème :



b) 10.8% des participants seront disqualifiés dans les conditions initiales.
c) A l'aide de notre réseau, si nous mettons le pourcentage de participants sain à 100% on peut en déduire qu'un participant sain a 1% de chance d'être disqualifié.
De la même manière si on met à 100% la probabilité d'être disqualifié on peut constater que 8.29% de participants disqualifiés sont sains.

2. Alarme incendie :

Voici le réseau bayésien modélisant le problème :



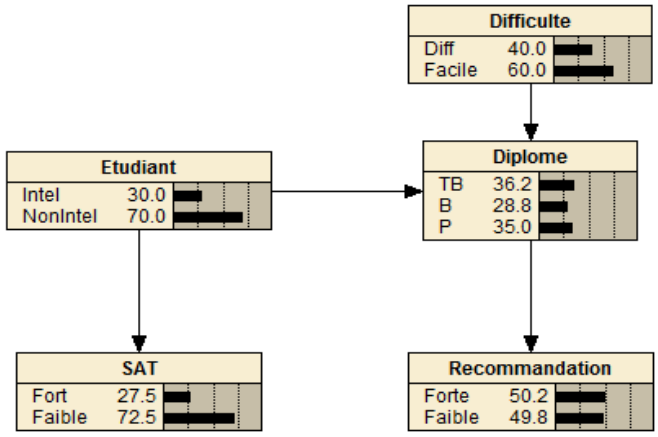
On considère qu'un opérateur est là 8h par jour soit 8/24h ce qui nous donne la probabilité qu'il déclenche l'alarme manuellement en cas d'incendie égale à 1/3 soit environ 33%.

En cas d'incendie la probabilité que l'alarme ne soit pas déclenchée est de 1.88%.

Si l'alarme ne s'est pas déclenchée en présence d'un incendie, on peut en déduire que l'opérateur n'était pas présent et que les détecteurs ont une probabilité de défaillance de 67.9%.

3. Diplôme avec mention :

1) La modélisation du réseau bayésien est la suivante :



2) A priori Georges a 50.2% de chance d'avoir une forte recommandation de son professeur. On apprend ensuite que Georges n'est pas une lumière, on met alors le pourcentage de NonIntel a 100% et on s'aperçoit que la probabilité pour qu'il ait une Recommandation positive est de 38.9%. On découvre ensuite qu'il a suivi une formation facile, on peut en déduire que ses chances d'avoir une mention sont plus élevé et donc la probabilité d'avoir une recommandation positive augmente à 51.3%.

3) Un autre étudiant a 30% de chance d'être intelligent selon la société. Après enquête elle découvre qu'il a eu passable, il a alors 7.89% de chances d'être intelligent. La formation a donc plus de chance d'être difficile, en effet la probabilité pour que la formation soit difficile est maintenant de 62.9%.
4) Si la recommandation de l'étudiant est faible, sa probabilité d'être brillant est de 14%.
5) Si le recruteur connaît à la fois la mention passable et la lettre de recommandation faible, ça revient au même que quand il ne connait que la mention car la lettre de recommandation est déterminée uniquement à l'aide de la mention de l'étudiant.
6) La probabilité que l'étudiant soit intelligent sans prendre en compte son score SAT est de 7.89%, si son score SAT est fort, la probabilité qu'il soit intelligent augmente à 57.8%. Ceci s'explique car l'étudiant a pu louper son examen ponctuel mais être intelligent quand même et donc réussir le test SAT.

IV. Nous contacter :

Si il y a des erreurs, des remarques, des ajouts à faire, etc.
Veuillez en faire part à une de ces adresses :

wedg@hotmail.fr (Veysseire Daniel)
mickaelfabre@free.fr (Fabre Mickael)