Function Filtrer (pattern, datum)

```
Function Filtrer (pattern, datum)
  if pattern est un symbol then
    if datum = pattern then return {}
    if pattern est un variable then return {pattern / datum}   ← return Echec
  if datum est un symbol then return Echec
  $F_1$ ← 1er elmt de datum, $T_1$ ← reste de datum
  $F_2$ ← 1er elmt de pattern, $T_2$ ← rest de pattern
  $Z_1$ ← Filtrer ($F_2$, $F_1$)
  if $Z_1$ = Echec then return Echec
  $G_1 = T_1$
  $G_2$ = remplacer les variables de $T_2$ par les unificateurs $Z_1$
  $Z_2$ = Filtrer ($T_2$, $T_1$)
  if $Z_2$ = Echec then return Echec
  return ($Z_1 \cup Z_2$)
```

Function Unifier (E_1, E_2)

```
Function Unifier ($E_1$, $E_2$)
  if $E_1$ ou $E_2$ est un symbole then
    Interchanger les argmts de $E_1$ et de $E_2$ (si nécessaire)
    de sorte que $E_1$ soit un atome
    if $E_1$ et $E_2$ sont identq. then return {}
    if $E_1$ est un variable then
      if $E_1$ apparait dans $E_2$ then return Echec
      return {$E_1$ : $E_2$}
    if $E_2$ est une variable then return {$E_1$ : $E_2$}
    return Echec
  $F_1$ ← head de $E_1$, $T_1$ ← tail de $E_1$
  $F_2$ ← head de $E_2$, $T_2$ ← tail de $E_2$
  $Z_1$ ← Unifier ($F_1$, $F_2$)
  if $Z_1$ = Echec then return Echec
  $G_1$ ← remplacer les vars de $T_1$ par les subst. $Z_1$
  $G_2$ ←   "        "      $T_2$    "        "     $Z_1$
  $Z_2$ ← Unifier ($G_1$, $G_2$)
  if $Z_2$ = Echec then return Echec
  return ($Z_1 \cup Z_2$)
```

Function Recherche-A* (Noeud-initial)

```
Function Recherche-A* (Noeud-initial)
Q ← (Noeud-initial)
C ← vide
repeat
  n ← first(Q), Q ← rest(Q)
  if n ∉ C, ou n ≡ n' ∈ C mais g(n) < g(n') then
    if n est un noeud but, return n
    ajouter n à C
    S ← succ(n)
    S ← sort (S, f)
    Q ← merge (S, Q, f)
until Q est vide
return Echec
```

Solution optimale garantie tant que h(n) sousestime le vrai coût.

PSC solveur par DFS :

X = tableau de n variables, rempli jusqu'à k
d = domaine des variables

```
Function DFS (x, d, k)
  if k ≥ n then
    return x
  else
    for v ∈ d[k+1] do
      consistent ← true
      for i ← 1 to k do
        if ¬consistent (v, x[i], C(i, k+1)) then
          consistent ← false
      if consistent then
        x [k+1] ← v
        rest ← DFS (x, d, k+1)
        if rest ≠ Echec then return rest
  return Echec
```

Backjumping : Quand il n'y a pas de valeur consistante pour la variable x_{k+1}, revenir directement à la dernière variable qui a une contrainte avec x_{k+1}

Conflict-directed-backjumping : Quand il n'y a pas de valeur consistante pour la var. x_{k+1}, revenir à la dernière variable qui avait un conflit avec une des valeurs du domaine de x_{k+1}

Forward checking : A chaque instanciation d'une variable x_k, éliminer toutes les valeurs inconsistantes avec x_k des labels des vars qui ne sont pas encore instanciées.

FC avec lookahead : si il y a des |label| = 1 alors propager, effacer v ∈ label des autres label de vars

DVO : ordonne en fonction de ||label||
→ combine avec FC/lookahead

min-width ordering : prendre la var non-inst. connectée avec le + petit # de var non instanciées

max-degree-ordering : prendre la var la + connecté (ne dépend pas du pas, statie)

→ ordonnancement des variables

Résultats :

- retour arrière simple = la moins efficace des méthodes
- backjumping (simple) est mieux, mais visite au moins autant de noeuds que FC et donc strictement moins efficace que FC
- FC et FC-lookahead = meilleurs heurist!
- conflict-dir-bj. peut ↗ perf du fc

Méthodes itératives :

- déterministe ; min-conflicts
- probabilist ; recuit simulé (= p. de changmt

Procedure chaînage-simple (F, R)

```
Procedure chaînage-simple (F, R)
Q ← {F}
while Q pas vide do
  q ← first(Q) ; Q ← rest(Q)
  if q ∉ base de données then
    ajouter q à la bdd
    if q est un noeud but then
      imprimer q
    for r ∈ {R} do
      if q ∈ cond (r) and
      cond(r) ⊆ bdd then
        ajouter concl(r) à Q
```

Procedure chaînage-Avant-Variables (F, R)

```
Procedure chaînage-Avant-Variables (F, R)
  Q ← faits de départ F
  while Q pas vide do
    q ← first (Q), Q ← rest (Q)
    if q ∉ bdd then
      imprimer q and ajouter q à bdd
    for r ∈ {R} do
      if ∃c ∈ conditions(r) and
      Filtrer (q, c) ≠ Echec then
        for toute combinaison C d'elmt
          de la bdd tq Filtrer (C, cond(r)) ≠ Echec do
          n ← instanciation de conclusion(r) avec
               les substitutions de Filtrer (C, cond(r))
          if n ∉ bdd, ajouter n en queue de Q
```

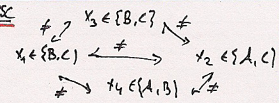
function Chaînage-Arrière (R, F, B)

```
function Chaînage-Arrière (R, F, B)
  envs ← {buts-initiaux} = B
  repeat
    e ← head (envs), envs ← rest (envs)
    if tous les but de e sont resolus then
      return instancier (buts(e), unificateurs (e))
    for tout but b ∈ e do
      for toute proposition p de la bdd F do
        if U ← Unifier (p, b) ≠ ECHEC then
          ajouter (e\b) ∪ U à la file envs
      for toute règle r de R do
        if U ← unifier (droit(r), b) ≠ ECHEC then
          ajouter (e\b) ∪ instancier (gauche(r), U)
             à la file envs
  until envs = vide
```

Function Recherche DFS BFS (Noeud-initial)

```
Function Recherche DFS BFS (Noeud-initial)
  Q ← (Noeud-initial)
  repeat
    n ← first (Q), Q ← rest (Q)
    if n est un noeud but, return n
    S ← succ(n)
    Q ← append (S, Q | Q, S)
  until Q est vide
  return Echec
```

appl. PSC



Domaines :
$D_1 \in \{B, C\}$, $D_2 \in \{A, C\}$, ...

Contraintes :
$C(X_1, X_2) : \{(B, A), (B, C), (C, A)\}$
$C(X_1, X_3) : \{(B, C)\ (C, B)\}$
$C(X_1, X_4) : \{(B, A)\ (C, B)\ (C, A)\}$
$C(X_2, X_3) : \{(A, B), (A, C), (C, B)\}$
$C(X_2, X_4) : \{(A, B)(C, A)(C, B)\}$

algo appelé avec d = [(B, C), (A, C), ...]

Trace :

Pas	k	X[1]	X[2]	X[3]	X[4]	retour arrière
1	1	B	–	–	–	
2	2	B	A	–	–	
3	3	B	A	C	–	
4	4	B	A	C	*	r. a!
5	3	B	A	*	–	r. a!
6	2	B	C	–	–	
7	3	B	C	*	–	r a!
8	2	B	*	–	–	r a!
9	1	C	–	–	–	
10	2	C	A	–	–	
11	3	C	A	B	–	
12	4	C	A	B	B	Solution

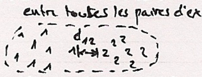
FC : [illegible] pas, 6 avec lookahead

Apprentissage non-supervisé :

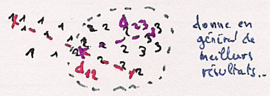
* Méthode par agglomération (dendogramme)

1) placer chaque exemple dans son propre cluster
2) trouver la paire de clusters la plus similaire (suivant la définition de distance entre les clusters) et la fusionner en 1 seul cluster
3) calculer la distance entre ce nouveau cluster et tous les autres clusters
4) répéter 2) et 3) jusqu'à ce qu'il n'y ait plus qu'un seul cluster contenant tous les exemple

- Single link : distance entre cluster = min des distances entre toutes les paires d'exemples



- complete link : maximum des distances



donne en général de meilleurs résultats..

* Méthode par division

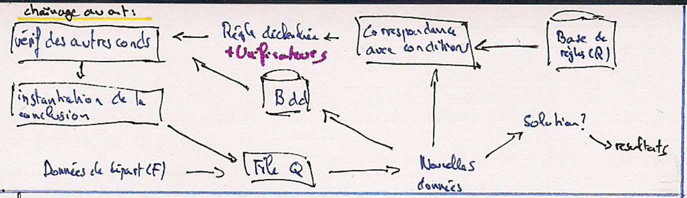
- On crée un MSTree des exemples existants
- On efface l'arrête avec la distance la plus grande. On itère sur cette étape jusqu'à un critère de fin

Clustering de partitionnement : K-means

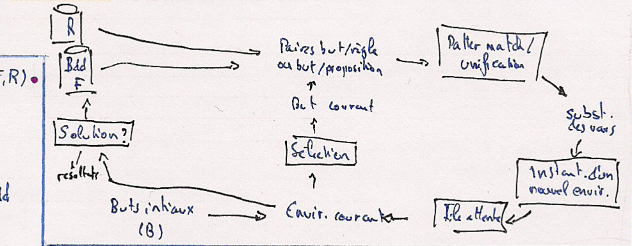
On veut k clusters. On choisit k noyaux c_j.
Ensuite, itération :

- associer à chaque exemple x_i à l'agrégat dont le noyau est le plus proche, càd c_j tq $d(c_j, x_i)$ est minimale ;
- Pour chaque agrégat C_j, remplacer le noyau par l'exemple qui est le plus au centre des exemples de l'agrégat, càd : $x_k \in C_j$ tq $\sum$ [illegible] $d(x_k, x_i)^2$ est min

chaînage avant :



chaînage arrière



P, Q, P → Q

déduction :
$$\frac{P \quad P \to Q}{Q}$$

abduction :
$$\frac{Q \quad P \to Q}{P}$$

induction :
$$\frac{P_1, P_2, \ldots \quad Q_1, Q_2, \ldots}{P \to Q}$$

! A souvent

Forme normale : Pour résol : f.n. conjonct.

- conj : $a_1 \wedge a_2 \wedge \ldots$
- disjonct. : $b_1 \vee b_2 \vee \ldots$

ex : $a_1 \wedge (a_2 \vee (a_3 \wedge a_4))$:
conj. $a_1 \wedge (a_2 \vee a_3) \wedge (a_2 \vee a_4)$
disj : $(a_1 \wedge a_2) \vee (a_1 \wedge a_2 \wedge a_3)$
pour résol : 3 clauses : $a_1\ (a_2 \vee a_3)\ (a_2 \vee a_4)$

Function DFS-cycle (noeud-initial)

```
Function DFS-cycle (noeud-initial)
  Q ← (Noeud initial)
  C ← vide
  repeat
    n ← first(Q), Q ← rest(Q)
    if n ∉ C then
      if n est un noeud but, return n
      ajouter n à C
      S ← succ(n)
      Q ← append(S, Q)
  until Q est vide
  return Echec
```

Probas :
$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(B|A)\,p(A)}{P(B)}$$
$$= \alpha \cdot p(B|A)\, p(A)$$
$$P(X|Y_1, \ldots, Y_k) = \alpha\, P(X) \prod_{i=1}^{k} p(Y_i|X)$$

$\textcircled{A} \to \textcircled{B} \to \textcircled{C}$: $p(B) \leftarrow \alpha\, \pi(B)\, \lambda(B)$ où
$\pi(B) = P(A|B)$, $\lambda(B) = P(B|C)$
$$P(B) = P(B|A, C) = P(A|B, C)\, \frac{P(B|C)}{P(A|C)}$$
$$= \underbrace{P(A|B)}_{\pi}\ \underbrace{p(B|C)}_{\lambda} \cdot \underbrace{\frac{1}{P(A|C)}}_{\alpha}$$

3 Manière d'implémenter un diagnostic :

1/ par abduction explicite : (A*)
recherche toute combinaison de câbles défectueux en retenant celles qui donne un modèle = défaut observé..
⚠ hyp. monde clos...

2/ Transformation en déduction ⚠ hyp. monde clos

$$\begin{array}{c|l} \text{obs} & \text{défaut} \\ \ldots & \{\} \\ & c_1 \wedge c_2 \end{array} \qquad \begin{array}{c|l} \ldots & (c_1 \wedge c_2) \vee (c_3 \wedge c_4) \end{array}$$
{ pas traduis. en clause de Horn !

3/ abduction par raisonnement incertain
Utilise un Facteur de Certitude (CF)
→ !
- choix CF
- aucune garantie
- need expérimentation !

Function Perceptron (P, N)

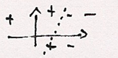
```
Function Perceptron (P, N)
  H ← (1, 0, 0, .., 0)   // vecteur initial aléatoire
  for $X_i$ ∈ P ∪ N do
    if C (H, $X_i$) ≠ class $X_i$ then   // si la classification ok,
                                         fait rien, sinon on change,
      if class ($X_i$) = + then   // trop petit
        H ← H + σ · $X_i$
      else   // trop grand
        H ← H − σ · $X_i$
  return H
```

Frontière linéaire :

$w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots w_n \cdot x_n \geq w_{n+1}$

fonction :
$\delta(w, x) = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n$

σ ≡ vitesse de convergence



! Si pas de frontière, pas de convergence vers un résultat !

Apprentissage de disjonctions

```
Function DisJonct (P, N)
  PSET ← P
  TERMS ← {}
  repeat
    D ← Description qui couvre X ⊆ PSet mais aucun
         n ∈ N
    PSet ← PSET − X
    TERMS ← TERMS ∪ D
  until PSet = {}
  return TERMS
```

Apprentissage avec exception

```
Function DL (E)
  DList ← {<classe par défaut>}
  repeat
    MISSED ← p ∈ P ∪ N pas correctement classifié par DList
    C ← Classe la + commune de MISSED
    D ← conjonction qui couvre le plus d'instances de
          C ∈ MISSED
    if D ≠ first (DLIST) then
      DLIST ← cons (D ⇒ C, DLIST)
    else
      selectionner un attribut A qui divise missed
          en $M_1$ et $M_2$
      recursion sur $M_1$ et $M_2$
  until MISSED = {}
  return DLIST
```

Function ID3 (E, A)

```
Function ID3 (E, A)
  if E = {} then return NIL
  else
    if ∀e ∈ E, classe(e) = c then
      return c
    else
      P ← attribut ∈ A qui réduit au plus l'entropie de la classe
      L ← {e | e ∈ E et P(e) = succès}
      R ← {e | e ∈ E et P(e) = echec}
      N ← noeud vide ; N.P ← P
      N.LEFT ← ID3 (L, A), N.RIGHT ← ID3 (R, A)
      return N
```

! Supervisé : trouver des règles de classification à partir d'exemples classifiés (c'est le cas d'ID3)

Formules :

- entropie de la classification : moyenne pondérée de tous les sous arbres du test
$$H(C|A) = \sum_{j=1}^{m} p(a_j) \cdot H(C|a_j) \quad \text{où } m = \#\text{ de valeurs pour l'attribut } A$$
- entropie du sous-arbre où A vaut a_j :
$$H(C|a_j) = -\sum_{i=1}^{n} p(c_i|a_j) \cdot \log_2 p(c_i|a_j) \quad \text{où } n = \#\text{ de classes différentes}$$
- entropie de la classification C :
$$H(C) = -\sum_{i=1}^{n} p(c_i) \cdot \log_2 p(c_i)$$

Algo génétique :

* entrée :
- population initiale de n solutions potentielles
- fonction d'évaluation f d'un solution ~ "sel. natur."

* Opérateurs de mutation et combinaison :
- mutation : change un élément du chr. de manière aléatoire
- combinaison : nouveau chr. en combinant des parties de 2 chr. → empêche "min. locaux"

Génération suivante construite à partir de :
- k solutions dont f est la + élevée (sel. naturelle)
- (n-k) sol. choisies au hasard.

On peut aussi favoriser la diversité : $f(s) + g(s)$ où $g(s) = \dfrac{1}{\sum_i \frac{1}{d^2(s, s_i)}}$

Clustering flou : exemple peut ∈ à plusieurs agrégats à la fois (probabilité d'appartenance)

Inférence par résolution :

1) BD ← {P}
2) Select. 2 clauses ∈ BD :
$P_1 : a_1 \vee \ldots \vee a_n \vee C$
$P_2 : b_1 \vee \ldots \vee b_m \vee \neg C$
3) appliquer la règle de résolution pour avoir :
$q : a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m$
4) si q ∉ BD, l'ajouter
5) si q solution, imprimer q (et arrêt)
6) répéter depuis 2)

- fondée (sol. tj valable)
- complète pour la réfutation

Preuve par ⊥ :

$(\{P\} \cup \neg q) \vdash \bot$

- prouve que q est une conséquence de {P}
- ⊥ peut ne pas s'arrêter si q n'est pas conséquence de {P}

Clause de Horn : au plus 1 proposition positive

$cond_1 \wedge cond_2 \wedge cond_3 \ldots \Rightarrow conséqu$
$\equiv \neg(cond_1 \wedge cond_2 \wedge \ldots) \vee conséqu$
$\equiv \neg cond_1 \vee \neg cond_2 \vee \ldots \vee conséqu$

donc $q \vee \neg p_1 \vee \neg p_2 \vee \neg \ldots$
$\equiv \{p_1, p_2\} \models q$

Chaînage sur Cl. de Horn : $\{P\} \cup \{\neg q\} \Rightarrow \bot$ — q : spec de la sol, aucune prop. nég. !

- avant : modus ponens pour produire q et ainsi avoir ⊥
- arrière : résol. de ¬q avec règle ou fait, ensuite applic. récursive sur les cond. de la règle

NB : résol de cl. de Horn → cl. de Horn :
Seule contradiction possible entre cl. de H, X et ¬X ⇒ procédure simplifiée

$$\frac{\neg \ldots \vee \neg \ldots \vee \neg X \vee Y \qquad \neg \ldots \vee \neg \ldots \vee X}{\neg \ldots \vee \neg \ldots \vee Y}$$

Traduction d'un problème de Planification en PSC (SAT pour être précis)

Repose sur 2 hypothèses:

- On connaît la longueur maximale ℓ du plan, càd le nombre maximal d'états. ⇝ iterative deepening
- On a déterminé toutes les actions (par avance) qui pourraient être nécessaire pour obtenir un plan d'action complet.

Description:

- Séquence d'états $S_1, S_2, \ldots, S_{\ell+1}$ avec $S_{\ell+1}$, état final qui n'admet plus aucune action.
- Pour chaque état, le PSC contient 2 types de variables:
  - * Pour chaque action qui pourrait faire partie du plan, il existe une variable qui prendra la valeur vrai si l'action est exécutée dans cet état et faux si elle ne l'est pas.
  - * Pour chaque propriété qui caractérise les états dans le formalisme STRIPS, il existe une variable qui indique si la propriété est vraie ou fausse dans cet état.

Contraintes:

- variable qui représente une action a des contraintes avec chaque variable qui représente une précondition dans le même état, càd action ⇒ précondition
- analogue pour les postconditions et les suppressions de chaque action
- condition initiales
- buts (= contraintes sur les variables du dernier état)
- axiomes de cadres également exprimés sous forme de contraintes (= si var. d'état pas touchée par une action, alors sa valeur au prochain état doit rester inchangée)
- mutex entre toutes paires d'actions tel que l'un influence la précondition de l'autre.

  expl: a_1 a précond. p / a_2 a postcond $\neg p$ |→ risque de conflit si exécutées simultanément

Expl. Planif avec PSC (missionnaires)

- 3 types d'acteurs: B; M_1 et M_2; C_1 et C_2
- 2 types "d'actions": traversées g.à d. et d.à g.
- Propositions du Probl. de Planif: $g(A)$, $d(A)$ où A acteur.
- Opérateurs du problème de Planif:

  $gd(B,M,A)$ où $A \in \{M_i, C_i\}$

  $dg(B,M)$ inutile de permettre C de revenir... (évite les "boucles")
- Conditions initiales:

  $g(B) = g(M_1) = g(M_2) = g(C_1) = g(C_2) = \text{True}$
- Conditions finales:

  $d(B) = d(M_1) = d(M_2) = d(C_1) = d(C_2) = \text{True}$
- Mutex de propositions:

  $[\, g(A), g(A) \,]$ pour tout A (acteur)
- Mutex d'opérateurs

  Pour chaque paire d'opérateurs distincts op_1 et op_2, si les deux opérateurs ont un acteur en commun, on a le mutex: $[op_1, op_2]$

  qui traduit le fait qu'un bateau ne peut pas contenir deux équipages différents en même temps, et aussi qu'un même M ou C ne peut se trouver sur deux bateau différents en même temps.

Plan non linéaire est constitué d'une séquence d'états $S_0, S_1, \ldots, S_n$ où chaque S_i est défini par des propositions qui sont vraies ou fausses au début de l'état, un ensemble d'opérat. qui sont exécutés pendant cet état, et des propositions qui sont vraies ou fausses à la fin de cet état.

Formalisme de STRIPS:

OPERATEUR (X):

PRECONDITIONS (P) = ...
DELETE (D) = ...
ADD (A) = ...

Choix des variables:

$\forall S_i$,
- $g(A, S_i) = \text{True} \Leftrightarrow$ A est sur la rive gauche à la fin de S_{i-1} / au début de S_i
- $d(A, S_i) = \text{True} \Leftrightarrow$ " " droite " " "

Contraintes (6 types!)

1) Contraintes sur l'état initial: $g(B, S_0) = g(M_1, S_0) = g(M_2, S_0) = g(C_1, S_0) = g(C_2, S_0) = \text{True}$
2) Contraintes sur l'état final: On "devine" le nbr d'états requis...

   $d(B, S_5) = d(M_1, S_5) = d(M_2, S_5) = d(C_1, S_5) = d(C_2, S_5) = \text{True}$
3) Contraintes pré/post-conditions des opérateurs:

   Pour chaque état S_i avec $i = 0 \ldots n$ et chaque opérateur op:
   - pour chaque proposition prop qui est une postcondition de l'opérateur op: $op(S_i) \Rightarrow Prop(S_i + 1)$
   - pour chaque proposition prop qui est une précondition de l'opérateur op: $op(S_i) \Rightarrow prop(S_i)$
4) contraintes PSC corresp. aux axiomes de cadres:

   Pour chaque proposition prop:
   - si $prop(S_i) = \text{False}$ et $prop(S_{i+1}) = \text{True}$, alors pour au moins un opérateur op qui a prop comme postcondition, on a $op(S_i) = \text{True}$
   - si $prop(S_i) = \text{True}$ et $prop(S_{i+1}) = \text{False}$, " " ... qui a prop comme postcondition négative, on a $op(S_i) = \text{True}$
5) mutex de propositions: $\forall S_i$, $i = 1..n$

   $g(A, S_i)$ NAND $d(A, S_i)$
6) mutex d'opérateurs: de la même manière que pour les mutex de propositions: $op_1(S_i)$ NAND $op_2(S_i)$

Exemple ID3

- donnée:
  - a) grand, allongé, rouge, piquant
  - b) petit, allongé, rouge, piquant
  - c) petit, rond, vert, piquant
  - d) grand, rond, vert, ¬ piquant
  - e) petit, allongé, jaune, ¬ piquant
  - f) petit, rond, rouge, ¬ piquant

	1	0
P(rouge)	0.5	0.5

$Pr(piq \mid rouge) = 0.66$
$P(\neg piq \mid rouge) = 0.33$
$P(piq \mid \neg rouge) = 0.33$
$P(\neg piq \mid \neg rouge) = 0.66$

- incertitude sur la classification qui reste dans le cas où rouge est vrai: $H(C \mid rouge = vrai) =$

$$- \sum_{v \in \{piq, \neg pik\}} Pr(v \mid rouge = vrai) \cdot \log_2(Pr(v \mid rouge = vrai))$$

$$= -0.66 \log_2(0.66) - 0.33 \log_2(0.33)$$

$$= 0.92 \text{ bit}$$

même résultat pour $H(C \mid rouge = faux)$

- entropie moyenne → moyenne pondérée

$$H(C \mid rouge) = H(C \mid rouge = vrai) \cdot Pr(rouge = vrai) + H(C \mid rouge = faux) \cdot Pr(rouge = faux)$$

$$= 0.5 \cdot 0.92 + 0.5 \cdot 0.92 = 0.92 \text{ bit}$$

...

Ce qui donne les étapes:

1) a, b, c, d, e, f

   grand, petit ou vert: incert. moyenne restante: 1 bit
   rond, allongé ou rouge: 0.92 bit
   jaune: 0.81 bit ⇒ choisir jaune
2) a, b, c, d, f

   grand ou petit: 0.955 bit
   rond ou allongé: 0.554 bit
   rouge ou vert: 0.955 bit ⇒ choisir rond
3) c, d, f

   grand ou petit: 0.66 bit
   rouge ou vert: 0.66 bit ⇒ choisir vert (au hasard)
4) c, d

   grand ou petit: 0 bit ⇒ petit

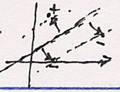
Diagnostique: Quels sont les candidats valables?

MS = Modèle du Système

- Comportement défaillant: $MS \cup OBS \vdash \bot$
- 2 versions:
  - le candidat explique les observations: (diagn. médical p.ex) $MS \cup CAND \vdash OBS$
  - le candidat rend les observations consistantes: (diag par consistance) $(MS - CAND) \cup OBS \nvdash \bot$ (diag. auto ↳ pot faulty ↳ changer le pot)

SVM: maximiser la séparation des exemples ⇒ choisir w tq:

- toutes les instances positives: d $\delta > \delta_0$
- " ... " négatives: $\delta < -\delta_0$
- δ_0 est maximale



Théorie PAC (chiffrer la qualité qu'on peut espérer d'un résultat d'apprentissage en fn de sa complexité et # d'exemples fournis

- Probablement: classif. approx. correcte avec proba δ
- Approximativement: proba d'erreur de classif. est $< \epsilon$
- Correct

On suppose:
- N exemples, concept a apprendre parmi |P| possibilités
- algo donne résultat correct sur tous les exples fournis pour l'apprentissage.
- $h \in H$ a un taux d'erreur $> \epsilon$

Proba que h soit correct sur les N exemples:

$$Pr(Correct(h, N)) < (1 - \epsilon)^N$$

PAC est satisfaite ⇔ Proba qu'il ∃ un h qui est correct sur N exemples mais a un taux d'erreur > ε sur de nouveaux exemples est < δ: $\delta > |P| \cdot (1 - \epsilon)^N$

Limite sur N pour borne δ_0 sur δ:

$$N \geq \frac{\ln(\delta_0 / |P|)}{\ln(1 - \epsilon)}$$

Exemple pour domaine où 10 attributs: il y a au plus $|P| = 11 \cdot 10^2 \cdot 9^4 \cdot 8^8 \cdots 2^{29} = 2{,}6579 \cdot 10^{35}$ différents arbres possibles.

Donc si on veut $\epsilon < 0.001$ avec proba $\delta < 0.01$, on doit avoir $N \geq \frac{\ln(\delta / |P|)}{\ln(1-\epsilon)} = 86\,131$ exemples

Si on se contente de $\epsilon \leq 0.05$: 1680 exemples

Méthode de taillage de l'arbre pour contrer l'overfitting

Bagging et Boosting:

Bagging: appliquer un certain # n de méthodes d'apprentissage faible, choisies de manière aléatoire, et prendre un vote parmi les classificateurs: si au moins k sur n donnent un résultat positif, alors conclut avec classif. positive, sinon négative.

Boosting: Construit k classification simples en choisissant chaque fois une distribution de proba des exemples qui met l'accent sur les exemples qui sont mal classés par les classifications déjà apprises.

- Initialement, tous les n exemples ont le même poids $w_i = 1/n$. L'algo est une itération pour $t \in 1 \ldots k$:

```
1) for $i \in 1..n$, $p_i \leftarrow \frac{w_i}{\sum_{i=1}^n w_i}$
2) $h_i \leftarrow$ résultat de la méthode faible
       avec les exemples et proba $p_i$
3) erreur $\epsilon \leftarrow \sum_{i=1}^n p_i |h(x_i) - c_i|$
4) $\beta_t \leftarrow \frac{\epsilon}{(1-\epsilon)}$
5) for $i \in 1...n$, $w_i \leftarrow w_i \beta^{1-|h(x_i)-y_i|}$
```

$$h_f(x) = \begin{cases} 1 & \text{si } \sum_{j=1}^{k} (-\log \beta_j)(h_j(x) - 1/2) \geq 0 \\ 0 & \text{sinon} \end{cases}$$

↳ pour combiner les différents classificateurs.

En reprenant la donnée d'id3. On suppose qu'on utilise un seul attribut qui prédit piquant ou ¬piquant avec fiabilité ≥ 50%

3 classifications faibles:

1) h_1: allongé ⇒ piquant

erreurs	ϵ_1	β_1	w_1	w_2	w_3	w_4	w_5	w_6
3, 5	1/3	1/2	1/12	1/12	1/6	1/12	1/6	1/12

$\Rightarrow P = \left(\frac{1}{8}, \frac{1}{8}, \frac{1}{4}, \frac{1}{8}, \frac{1}{4}, \frac{1}{8}\right)$

2) h_2: jaune ⇒ ¬piquant

erreurs	ϵ_2	β_2	w_1	w_2	w_3	w_4	w_5	w_6
6	1/8	1/7	1/84	1/84	1/42	1/84	1/42	1/12

$\Rightarrow P = \left(\frac{1}{14}, \frac{1}{14}, \frac{1}{7}, \frac{1}{14}, \frac{1}{7}, \frac{1}{2}\right)$

3) h_3: petit ⇒ piquant

erreurs	ϵ_3	β_3	w_1	w_2	w_3	w_4	w_5	w_6
5	1/7	1/6	1/84	1/84	1/42	1/84	1/42	1/12

Le résultat de la classification est donc:

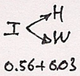
$$h_f(x) = \log(2) h_1(x) + \log(7) h_2(x) + \log(6) h_3(x) - \frac{\log(2) + \log(7) + \log(6)}{2}$$

$$= 0.69 h_1(x) + 1.95 h_2(x) + 1.8 h_3(x) - 2.22$$

où $h_i(x)$ valent 0 ou 1 suivant le cas

Traitement de l'information incertaine:

I = Route gelée, H = accident Holmes, W = accident-Watson



$P(I=1) = 0.7$
$P(H=1 \mid I=1) = P(W=1 \mid I=1) = 0.8$
$P(H=1 \mid I=0) = P(W=1 \mid I=0) = 0.1$

a)

P(H|I)	I=1	I=0
H=1	0.8	0.1
H=0	0.2	0.9

idem pour P(W\|I) et P(W,I)

P(H,I)	I=1	I=0
H=1	0.8·0.7 = 0.56	0.1·0.3 = 0.03
H=0	0.2·0.7 = 0.14	0.9·0.3 = 0.27

b) Construction de P(H) et P(W) par le processus de marginalisation:

$P(H=1) = 0.56 + 0.03 = 0.59$
$P(H=0) = 0.41$
$P(W=1) = 0.59$
$P(W=0) = 0.41$

c) Proba que les routes soient gelées sachant que Watson a eu un accident:

$$P(I=1 \mid W=1) = \frac{P(W=1 \mid I=1) \cdot P(I=1)}{P(W)} = \frac{0.8 \cdot 0.7}{0.8 \cdot 0.7 + 0.1 \cdot 0.3} = \frac{0.56}{0.59} \simeq 0.95$$

d) Sachant que Watson a eu un accident, P(H)?

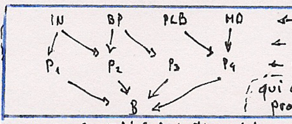
P(H,I)	I=1, W=1	I=0, W=1
H=1	0.8·0.95 = 0.76	0.1·0.05 = 0.005
H=0	0.2·0.95 = 0.19	0.9·0.05 = 0.045

→ $P(H) = 0.765 \approx 76{,}5\%$

et donc P(H) a augmenté!

e) Après que l'on ait appris que les routes ne sont pas gelées, on revient sur notre décision, pourquoi?

Car l'accident de Watson n'a plus d'influence sur la probabilité d'accident de Holmes. Les événements (2) sont devenus indépendants!



← défaillance possible
← peut toucher
← chemin

3 cycles qui empêche la propag. des proba.

1/ Stock de bon Bourgogne pour l'année 2007? Bonifier (Bourgogne, ?annee-vin, 2007): Quel vin éliminer de la cave? Eliminer (?Vin, ?Annee-Vin)

Théorie de Dempster et Schafer.

Evénement: Proba ∈ [0;1].
La limite inférieure est appelée la crédibilité
La limite supérieure est appelée la plausibilité

expl: pluie ⇒ mouillé.
crédibilité de mouillé au moins égal à P(pluie).
Cependant, P(mouillé) peut être bcp plus élevé si bcp de causes.
Plausibilité affectée par les contre-indications à mouillé.

sec ⇒ ¬mouillé.
donc plausibilité ≈ 1 − pr(sec)

Si causes/ évén. pas indép devient très vite complexe!

Function recuit(X, V, C)

```
V ← assignation aléatoire
T[max-steps] ← plan de réduction de "température"
for j=1 to max-steps do
    nconf ← check (V,C)
    if nconf=0 then return V
    V' ← V avec valeur V[k] changée aléatoirement
    if check(V',C) < check(V,C) then
        V ← V'
    else if random (0..1) < T[j]
        then V ← V'
return solution partielle V.
```

Par exple: T = (0.7, 0.5, 0.3, 0.1, 0.001)

→ permet d'éviter le problème des minimum locaux...

Proposition de mesure

Entropie de X étant donnée la mesure de Y: $E(X \mid Y) = \sum_j P(y_j) \left\{ \sum_i - P(x_i \mid y_j) \log P(x_i \mid y_j) \right\}$

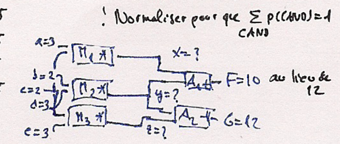
Probabilité de mesurer la valeur k pour la variable i:

$$p(var_i = val_{ik}) = \sum_{c \in P_{ik}} p(c) + \sum_{c \in U_i} p(c)/m$$

où P_{ik}: ensemble des candidats qui prédisent la valeur k pour la variable i
U_i: candidats qui ne donnent aucune valeur pour la mesure i

*Candidat	Prévision	Proba	Normalisé
{M1}	xyz = (4,6,6)	0.001	0.495
{A1}	xyz = (6,6,6)	0.001	0.495
{M2, A2}	xyz = (6,4,6)	10^{-6}	0.005
{M2, M3}	xyz = (6,4,8)	10^{-6}	0.005

! Normaliser pour que $\sum_{CAND} p(CAND) = 1$



et donc les proba. des différentes valeurs:

Mesure	Justification	Candidats	Probabilité
X=4	({M2, A1}, {M3, A1, A2})	{M1}	0.495
X=6	({M1})	{A1}, {M2, A2}, {M2, M3}	0.505
Y=6	({M2}, {M3, A2})	{M1}, {A1}	0.99
Y=4	({M1, A1})	{M2, A2}, {M2, M3}	0.01
Z=6	({M3}, {M2, A2})	{M1}, {A1}, {M2, A2}	0.995
Z=8	({M1, A1, A2})	{M2, M3}	0.005

où il faut observer qu'un candidat prédit une mesure si au moins une des justifications de la mesure ne contient aucun élément qui fait partie du candidat.

On obtient les entropies:

$X: -P(6)\log(P(6)) - P(4) - \log(P(4)) = -0.505 \log(0.505) - 0.495 \log(0.495) = 0.99\ldots$ bit
$Y: -P(6)\log(P(6)) - P(4)\log(P(4)) = 0.0808$ bit
$Z: -P(6)\log(P(6)) - P(8)\log(P(8)) = 0.0454$ bit

Et donc on décide de mesurer X

* On suppose que P(panne) = 0.001 pour tous les composants!

0e ordre: vole(TiH)
1er ordre: quant. symboles $\forall(x)$ oiseau(x) ⇒ vole(x)
2e ordre: quantif sur symbole et prédicat $(\forall(p))\ p(Charles) \Rightarrow p(\text{[illegible]})$
. IA seul 0e et 1er