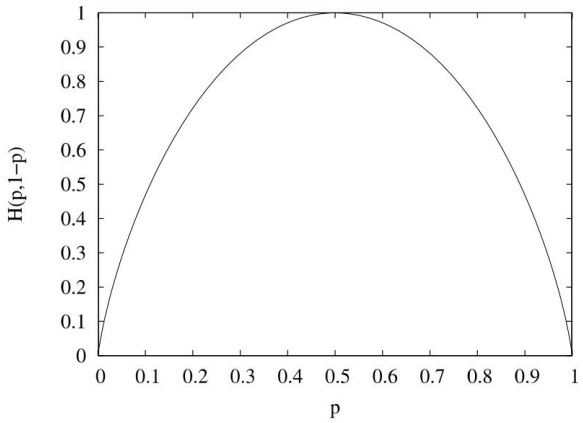


Fig. 8.24 La fonction d'entropie pour le cas de deux classes. Nous voyons le maximum à p = 1/2 et la symétrie par rapport à l'échange de p et 1 − p

Définition 8.4 L' entropie H en tant que métrique de l'incertitude d'une distribution de probabilité est définie par[7]

$$H(p) = H(p_1,\ldots,p_n) := -\sum_{i=1}^{n} p_i \log_2 p_i .$$

Une dérivation détaillée de cette formule se trouve dans [SW76]. Si nous substituons le certain événement p = (1, 0, 0,..., 0), alors 0 log2 0, une expression indéfinie en résulte. On résout ce problème par la définition 0 log2 0 := 0 (voir Exercice 8.9 page 218).

Nous pouvons maintenant calculer H (1, 0,..., 0) = 0. Nous allons montrer que l'entropie dans le hypercube [0, 1] ^{n} sous la contrainte $\sum_{i=1}^{n} p_i = 1$ prend sa valeur maximale). avec la distribution uniforme $\frac{1}{n},\ldots,\frac{1}{n}$ Dans le cas d'un événement avec deux possibilités (résultats, qui correspondent à deux classes, le résultat est

$$H(p) = H(p_1, p_2) = H(p_1, 1 - p_1) = -(p_1 \log_2 p_1 + (1 - p_1)\log_2 (1 - p_1)).$$

Cette expression est représentée en fonction de p1 sur la figure 8.24 avec son maximum à p1 = 1/2.

Étant donné que chaque ensemble de données classifié D se voit attribuer une distribution de probabilité p en estimant les probabilités de classe, nous pouvons étendre le concept d'entropie aux données en

[7] Dans (7.9) à la page 124, le logarithme naturel plutôt que log2 est utilisé dans la définition de l'entropie. Parce qu'ici, et aussi dans le cas de la méthode MaxEnt, les entropies ne sont que comparées, cette différence ne joue aucun rôle. (voir Exercice 8.11 à la page 218).

la définition

$$H(D) = H(p).$$

Maintenant, puisque le contenu d'information I (D) de l'ensemble de données D est censé être le contraire de l'incertitude. Ainsi nous définissons :

Définition 8.5 Le contenu informationnel d'un jeu de données est défini comme

$$Je(D) := 1 - H(D). \tag{8.6}$$

8.4.3 Gain d'informations

Si nous appliquons la formule d'entropie à l'exemple, le résultat est

$$H(11/06, 11/05) = 0{,}994$$

Lors de la construction d'un arbre de décision, l'ensemble de données est ensuite subdivisé par chaque nouvel attribut. Plus un attribut augmente le contenu informationnel de la distribution en divisant les données, meilleur est cet attribut. Nous définissons ainsi :

Définition 8.6 Le gain d'information G(D, A) grâce à l'utilisation de l'attribut A est déterminé par la différence entre le contenu moyen en information de l'ensemble de données D = D1 D2 ··· Dn divisé par l'attribut de valeur n A et le contenu informatif I (D) de l'ensemble de données non divisé, ce qui donne

$$G(D, A) = \sum_{je=1}^{n} \frac{|Di|}{|D|} Je(Di) - Je(D).$$

Avec (8.6) on obtient de cette

$$\begin{aligned} G(D, A) &= \sum_{je=1}^{n} \frac{|Di|}{|D|} je(Di) - je(D) = \sum_{je=1}^{n} \frac{|Di|}{|D|} (1 - H(Di)) - (1 - H(D)) \\ &= 1 - \sum_{je=1}^{n} \frac{|Di|}{|D|} H(Di) - 1 + H(D) \\ &= H(D) - \sum_{je=1}^{n} \frac{|Di|}{|D|} H(Di). \end{aligned} \tag{8.7}$$

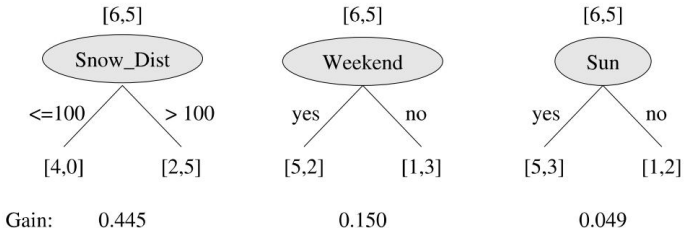


Fig. 8.25 Le gain calculé pour les différents attributs indique si la division des données par l'attribut respectif aboutit à une meilleure division de classe. Plus les distributions générées par l'attribut s'écartent de la distribution uniforme, plus le gain d'information est élevé

Appliqué à notre exemple pour l'attribut Snow_Dist, cela donne

$$G(D, \text{Snow\_Dist}) = H(D) - \frac{4}{11} H(D \le 100) + \frac{7}{11} H(D > 100)$$

$$= 0{,}994 - \frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0{,}863 = 0{,}445.$$

De manière analogue on obtient

$$G(J,\text{Week-end}) = 0{,}150$$

et

$$G(D, \text{Soleil}) = 0{,}049.$$

L'attribut Snow_Dist devient maintenant le nœud racine de l'arbre de décision. La situation de la sélection de cet attribut est à nouveau clarifiée dans la Fig. 8.25.

Les deux valeurs d'attribut ≤100 et >100 génèrent deux arêtes dans l'arbre, qui correspondent aux sous-ensembles D≤100 et D>100. Pour le sous-ensemble D≤100 la classification est clairement oui. ainsi l'arbre se termine ici. Dans l'autre branche D>100 il n'y a pas de résultat clair. Ainsi, l'algorithme se répète récursivement. Parmi les deux attributs encore disponibles, Sun et Weekend, il faut choisir le meilleur. Nous calculons

$$G(J>100,\text{Week-end}) = 0{,}292$$

et

$$G(D>100, \text{Soleil}) = 0{,}170.$$

Le nœud obtient ainsi l'attribut Weekend attribué. Pour Weekend = no , l'arborescence se termine par la décision Ski = no. Un calcul du gain renvoie ici la valeur 0.
Pour Weekend = yes, Sun donne un gain de 0,171. Ensuite, la construction de l'arbre se termine car aucun autre attribut n'est disponible, bien que l'exemple numéro 7 soit faussement classé. L'arbre fini est déjà familier de la Fig. 8.23 à la page 186.

8.4.4 Application de C4.5

L'arbre de décision que nous venons de générer peut également être généré par C4.5. Les données d'entraînement sont enregistrées dans un fichier de données ski.data au format suivant :

```
<=100, oui, oui, oui <=100, oui, oui,
oui <=100, oui, non, oui <=100, non,
oui, oui >100, oui, oui, oui >100, oui,
oui , oui >100, oui, oui, non >100,
oui, non, non >100, non, oui, non
>100, non, oui, non >100, non,




                    non non
```

Les informations sur les attributs et les classes sont stockées dans le fichier ski.names (les lignes commençant par « | » sont des commentaires) :

```
 |Cours : non : ne skie pas, oui : skie |

non Oui.
 |
 |Attributs |

Snow_Dist :              <=100,>100.
Fin de semaine:          non Oui.
Soleil:                  non Oui.
```

C4.5 est ensuite appelé à partir de la ligne de commande Unix et génère l'arbre de décision illustré ci-dessous, qui est formaté à l'aide d'indentations. L'option -f correspond au nom du fichier d'entrée et l'option -m spécifie le nombre minimum de points de données d'apprentissage requis pour générer une nouvelle branche dans l'arborescence. Étant donné que le nombre de points de données d'apprentissage dans cet exemple est extrêmement petit, -m 1 est raisonnable ici. Pour les jeux de données plus volumineux, une valeur d'au moins -m 10 doit être utilisée.

```
invite unix> c4.5 -f ski -m 1

Générateur d'arbre de décision C4.5 [version 8]

                                                    Mer 23 août 10:44:49 2010
----------------------------------------

        Option :
                Fichier tige <ski>
                Le test sensible nécessite 2 branches avec >= 1 cas

Lire 11 cas (3 attributs) de ski.data

Arbre de décision:

Snow_Dist = <=100 : oui (4.0)
Snow_Dist = >100 :
 | Week-end = non : non (3.0)
 | Week-end = oui :
 | | Soleil = non : non (1.0)
 | | Soleil = oui : oui (3.0/1.0)

Arbre de décision simplifié :

Snow_Dist = <=100 : oui (4.0/1.2)
Snow_Dist = >100 : non (7.0/3.4)

Évaluation sur les données d'entraînement (11 items) :

        Avant la taille                 Après la taille
        ---------------                 ---------------------------
        Taille      les erreurs         Taille      les erreurs     Estimation

          7         1(9,1 %)              3         2(18,2 %)       (41,7%) <<
```

De plus, un arbre simplifié avec un seul attribut est donné. Cet arbre, qui a été créé par l'élagage (voir Section 8.4.7), sera important pour une quantité croissante de données d'entraînement. Dans ce petit exemple cela n'a pas encore beaucoup de sens. L'erreur taux pour les deux arbres sur les données de formation est également donné. Les chiffres entre parenthèses après les décisions, donnez la taille de l'ensemble de données sous-jacent et le nombre d'erreurs. Par exemple, la ligne Soleil = oui : oui (3.0/1.0) dans l'arborescence du haut indique que pour ce nœud feuille Sun = yes, il existe trois exemples d'apprentissage, dont l'un est faussement classé. L'utilisateur peut ainsi lire si la décision est statistiquement fondée et/ou certaine.

Dans la Fig. 8.26 page 193 nous pouvons maintenant donner le schéma de l'algorithme d'apprentissage pour générer un arbre de décision.

Nous connaissons maintenant les fondements de la génération automatique de décisions des arbres. Pour l'application pratique, cependant, des extensions importantes sont nécessaires. Nous les introduira à l'aide de l' application LEXMED déjà familière .

```
GENERATEDECISIONTREE(Data,Node)
Amax = Attribut avec gain d'information maximal Si G(Amax ) = 0
Alors le nœud
    devient le nœud feuille avec la classe la plus fréquente dans les données Sinon
    affectez l'attribut Amax au nœud Pour chaque
        valeur a1,...,an de Amax , générez un nœud successeur :
                                                K1,...,Kn
        Diviser Data en D1,...,Dn avec Di = {x    Data|Amax(x) = ai}
        Pour tout i    {1,...,n}
            Si tous les x    Di appartiennent à la même classe Ci
                Générer ensuite le nœud feuille Ki de classe Ci
                Sinon GENERATEDECISIONTREE(Di, Ki)
```

Fig. 8.26 Algorithme pour la construction d'un arbre de décision

8.4.5 Apprentissage du diagnostic d'appendicite

Dans le projet de recherche LEXMED, un système expert pour le diagnostic de l'appendicite a été développé en plus d'une base de données de données de patients [ES99, SE00]. Le système, qui fonctionne avec la méthode d'entropie maximale, est décrit dans la Sect. 7.3

Nous utilisons maintenant la base de données LEXMED pour générer un arbre de décision pour le diagnostic d'appendicite à C4.5. Les symptômes utilisés comme attributs sont définis dans le fichier app.names :

```
|Définition des classes et des attributs | |Classes 0=appendicite

négative | 1=appendicite positif 0,1. | |Attributs | Âge :
continu.




Sexe_(1=m___2=w) : 1,2.
Douleur_Quadrant1_(0=non__1=oui) : 0,1.
Douleur_Quadrant2_(0=non__1=oui) : 0,1.
Douleur_Quadrant3_(0=non__1=oui) : 0,1.
Douleur_Quadrant4_(0=non__1=oui) : 0,1.
Surveillance_locale_(0=non__1=oui) : 0,1.
Surveillance_généralisée_(0=non__1=oui) : 0,1.
Rebond_tendresse_(0=non__1=oui) : 0,1.
Pain_on_tapping_(0=non__1=oui) : 0,1.
```

```
Douleur_pendant_l'examen_rectal_(0=non__1=oui) : 0,1.
Temp_axial : continu.
Temp_rectal : continu.
Leucocytes : continu.
Diabète_sucré_(0=non__1=oui) : 0,1
```

Nous voyons que, outre de nombreux attributs binaires tels que les divers symptômes de la douleur, des symptômes continus tels que l'âge et la température de la fièvre se produisent également. Dans le fichier de données de formation suivant, app.data, dans chaque ligne un cas est décrit. Dans la première ligne se trouve un patient de sexe masculin de 19 ans souffrant de douleurs dans le troisième quadrant (en bas à droite, où se trouve l'appendice), les deux valeurs de fièvre 36,2 et 37,8 degrés Celsius, une valeur leucocytaire de 13400 et un diagnostic positif, que est, un appendice enflammé.

```
19,1,0,0,1,0,1,0,1,1,0,362,378,13400,0,1
13,1,0,0,1,0,1,0,1,1,1,383,385,18100 ,0,1
32,2,0,0,1,0,1,0,1,1,0,364,374,11800,0,1 18,2,0,0,1,1,0,0,0,0
,0,362,370,09300,0,0 73,2,1,0,1,1,1,0,1,1,1,376,380,13600,1,1
30,1,1,1,1,1,0,1 ,1,1,1,377,387,21100,0,1
56,1,1,1,1,1,0,1,1,1,0,390,?,14100,0,1 36,1,0,0,1
,0,1,0,1,1,0,372,382,11300,0,1
36,2,0,0,1,0,0,0,1,1,1,370,379,15300,0,1 33,1,0
,0,1,0,1,0,1,1,0,367,376,17400,0,1
19,1,0,0,1,0,0,0,1,1,0,361,375,17600,0,1 12
,1,0,0,1,0,1,0,1,1,0,364,370,12900,0,0

...
```

Sans entrer dans les détails de la base de données, il est important de mentionner que seuls les patients suspects d'appendicite à leur arrivée à l'hôpital et opérés par la suite sont inclus dans la base de données. Nous voyons dans la septième ligne que C4.5 peut également traiter les valeurs manquantes. Les données contiennent 9764 cas.

```
invite unix> c4.5 -f app -u -m 100

Générateur d'arbre de décision C4.5 [version 8]
                                                    Mer 23 août 13:13:15 2006
----------------------------------------

Lire 9764 cas (15 attributs) à partir de app.data

Arbre de décision:
```

```
Leucocytes <= 11030 :
 | Rebond_tendresse = 0 :
 | | Temp_rectal > 381 : 1 (135.9/54.2)
 | | Temp_rectale <= 381 :
 | | | Local_guarding = 0 : 0 (1453.3/358.9)
 | | | Local_guarding = 1 :
 | | | | Sexe_(1=m___2=w) = 1 : 1 (160,1/74,9)
 | | | | Sexe_(1=m___2=w) = 2 : 0 (286,3/97,6)
 | Rebond_tendresse = 1 :
 | | Leucocytes <= 8600 :
 | | | Temp_rectal > 378 : 1 (176.0/59.4)
 | | | Temp_rectale <= 378 :
 | | | | Sexe_(1=m___2=w) = 1 :
 | | | | | Local_guarding = 0 : 0 (110,7/51,7)
 | | | | | Local_guarding = 1 : 1 (160,6/68,5)
 | | | | Sexe_(1=m___2=w) = 2 :
 | | | | | Âge <= 14 : 1 (131,1/63,1)
 | | | | | Âge > 14 : 0 (398,3/137,6)
 | | Leucocytes > 8600 :
 | | | Sexe_(1=m___2=w) = 1 : 1 (429,9/91,0)
 | | | Sexe_(1=m___2=w) = 2 :
 | | | | Local_guarding = 1 : 1 (311.2/103.0)
 | | | | Local_guarding = 0 :
 | | | | | Temp_rectale <= 375 : 1 (125.4/55.8)
 | | | | | Temp_rectal > 375 : 0 (118,3/56,1)
Leucocytes > 11030 :
 | Rebond_tendresse = 1 : 1 (4300.0/519.9)
 | Rebond_tendresse = 0 :
 | | Leucocytes > 14040 : 1 (826,6/163,8)
 | | Leucocytes <= 14040 :
 | | | Pain_on_tapping = 1 : 1 (260,6/83,7)
 | | | Pain_on_tapping = 0 :
 | | | | Local_guarding = 1 : 1 (117,5/44,4)
 | | | | Local_guarding = 0 :
 | | | | | Temp_axial <= 368 : 0 (131.9/57.4)
 | | | | | Temp_axial > 368 : 1 (130.5/57.8)

Arbre de décision simplifié :

Leucocytes > 11030 : 1 (5767.0/964.1)
Leucocytes <= 11030 :
 | Rebond_tendresse = 0 :
 | | Temp_rectal > 381 : 1 (135,9/58,7)
 | | Temp_rectale <= 381 :
 | | | Local_guarding = 0 : 0 (1453.3/370.9)
 | | | Local_guarding = 1 :
 | | | | Sexe_(1=m___2=w) = 1 : 1 (160,1/79,7)
 | | | | Sexe_(1=m___2=w) = 2 : 0 (286,3/103,7)
 | Rebond_tendresse = 1 :
 | | Leucocytes > 8600 : 1 (984,7/322,6)
 | | Leucocytes <= 8600 :
```

```
. . . | Temp_rectal > 378 : 1 (176.0/64.3)
. . . | Temp_rectale <= 378 :
. . . . | Sexe_(1=m___2=w) = 1 :
. . . . | | Local_guarding = 0 : 0 (110,7/55,8)
. . . . | | Local_guarding = 1 : 1 (160,6/73,4)
. . . . | Sexe_(1=m___2=w) = 2 :
. . . . | | Âge <= 14 : 1 (131,1/67,6)
. . . . | | Âge > 14 : 0 (398,3/144,7)

Évaluation sur données d'entraînement (9764 items) :

Avant la taille Après la taille
———————
Erreurs de taille Estimation des erreurs de taille
37 2197(22,5%) 21 2223(22,8%) (23,6%) <<

Évaluation sur données de test (4882 items) :

Avant la taille Après la taille
———————
Erreurs de taille Estimation des erreurs de taille
37 1148(23,5%) 21 1153(23,6%) (23,6%) <<

 (a) (b) <-classé comme
--
758 885 (a) : classe 0
268 2971 (b) : classe 1
```

8.4.6 Attributs continus

Dans les arbres générés pour le diagnostic d'appendicite, il y a un nœud Leucocytes
> 11030 qui vient clairement de l'attribut continu Leucocytes
en fixant un seuil à la valeur 11030. C4.5 a donc fait un tribut binaire Leucocytes > 11030 à partir de l'attribut continu Leucocytes.
Le seuil ΘD,A pour un attribut A est déterminé par l'algorithme suivant :
pour toutes les valeurs v qui apparaissent dans les données d'apprentissage D, l'attribut binaire A>v est généré et son gain d'information est calculé. Le seuil ΘD,A est alors fixé à
la valeur v avec le gain d'information maximum, soit :

$$\Theta_{D,A} = \operatorname{argmax}_v \{G(D, A > v)\}.$$

Pour un attribut tel que le taux de leucocytes ou l'âge du patient, une décision basée sur
une discrétisation binaire est vraisemblablement trop imprécise. Néanmoins il n'est pas nécessaire de discrétiser plus finement car chaque attribut continu est testé sur chaque nouvelle génération
nœud et peut donc se produire de manière répétée dans un arbre avec un seuil ΘD,A différent. Ainsi on obtient finalement une très bonne discrétisation dont la finesse s'adapte au problème.

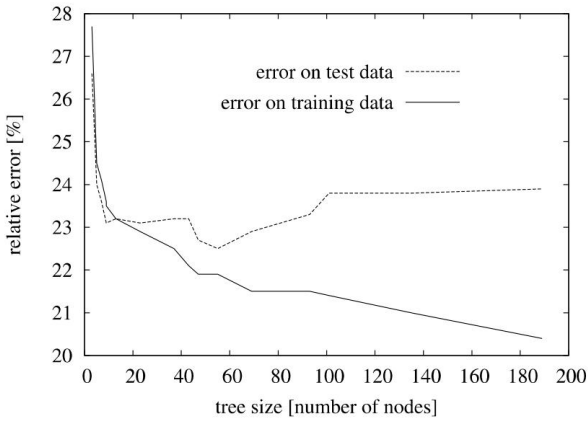


Fig. 8.27 Courbe d'apprentissage de C4.5 sur les données d'appendicite. On voit clairement le surajustement des arbres à plus de 55 nœuds

8.4.7 Élagage—Coupe de l'arbre

Depuis l'époque d'Aristote, il est stipulé que de deux théories scientifiques qui expliquent également bien la même situation, la plus simple est préférée. Cette loi de l'économie, également connue sous le nom de rasoir d'Occam, est d'une grande importance pour l'apprentissage automatique et l'exploration de données.

Un arbre de décision est une théorie permettant de décrire les données d'apprentissage. Une théorie différente pour décrire les données est les données elles-mêmes. Si l'arbre classe toutes les données sans aucune erreur, mais est beaucoup plus compact et donc plus facilement compréhensible par l'homme, alors il est préférable selon le rasoir d'Occam. Il en va de même pour deux arbres de décision de tailles différentes. Ainsi, le but de tout algorithme de génération d'arbre de décision doit être de générer le plus petit arbre de décision possible pour un taux d'erreur donné. Parmi tous les arbres avec un taux d'erreur fixe, le plus petit arbre doit toujours être sélectionné.

Jusqu'à présent, nous n'avons pas défini précisément le terme taux d'erreur. Comme déjà mentionné à plusieurs reprises, il est important que l'arbre appris ne se contente pas de mémoriser les données d'apprentissage, mais plutôt qu'il généralise bien. Pour tester la capacité d'un arbre à généraliser, nous divisons les données disponibles en un ensemble de données d'apprentissage et un ensemble de données de test. Les données de test sont cachées de l'algorithme d'apprentissage et ne sont utilisées que pour les tests. Si un grand ensemble de données est disponible, comme les données sur l'appendicite, nous pouvons par exemple utiliser les deux tiers des données pour l'apprentissage et le tiers restant pour les tests.

Outre une meilleure compréhensibilité, le rasoir d'Occam a une autre justification importante : la capacité de généralisation. Plus le modèle est complexe (ici un arbre de décision), plus les détails sont représentés, mais dans la même mesure moins le modèle est transférable à de nouvelles données. Cette relation est illustrée à la Fig. 8.27. Des arbres de décision de différentes tailles ont été entraînés par rapport aux données sur l'appendicite. Dans le graphique, les erreurs de classification sur les données de formation et sur les données de test sont données. Le taux d'erreur sur les données d'apprentissage diminue de manière monotone avec la taille de l'arbre. Jusqu'à une taille d'arbre de 55 nœuds, le taux d'erreur sur les données de test diminue également. Si l'arbre continue de croître, cependant, le taux d'erreur recommence à augmenter ! Cet effet, que nous avons déjà vu dans la méthode du plus proche voisin, est appelé surajustement.

Nous donnerons ce concept, qui est important pour presque tous les processus d'apprentissage, une définition générale tirée de [Mit97] :

Définition 8.7 Soit un algorithme d'apprentissage spécifique, c'est-à-dire un agent d'apprentissage. Nous appelons un agent A un surajustement aux données d'apprentissage s'il existe un autre agent A′ dont l'erreur sur les données d'apprentissage est supérieure à celle de A, mais dont l'erreur sur l'ensemble de la distribution des données est inférieure à l'erreur de A.

Comment pouvons-nous maintenant trouver ce point d'erreur minimum sur les données de test ? L'algorithme le plus évident est appelé validation croisée. Lors de la construction de l'arbre, l'erreur sur les données de test est mesurée en parallèle. Dès que l'erreur augmente de manière significative, l'arbre avec l'erreur minimale est enregistré. Cet algorithme est utilisé par le système CART mentionné précédemment.

C4.5 fonctionne un peu différemment. Tout d'abord, en utilisant l'algorithme GENERATEDECI SIONTREE de la Fig. 8.26 à la page 193, il génère un arbre qui est généralement surajusté. Ensuite, en utilisant l'élagage, il tente de couper des nœuds de l'arbre jusqu'à ce que l'erreur sur les données de test, estimée par l'erreur sur les données d'apprentissage, commence à augmenter8. Comme la construction de l'arbre, c'est aussi un algorithme gourmand. . Cela signifie qu'une fois qu'un nœud est élagué, il ne peut pas être réinséré, même si cela s'avère plus tard meilleur.

8.4.8 Valeurs manquantes

Souvent, des valeurs d'attribut individuelles manquent dans les données d'apprentissage. Dans l' ensemble de données LEX MED , l'entrée suivante apparaît :

56, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14100, 0, 1,

dans laquelle une des valeurs de fièvre est manquante. Ces données peuvent néanmoins être utilisées lors de la construction de l'arbre de décision. Nous pouvons attribuer à l'attribut la valeur la plus fréquente de l'ensemble de données ou le plus fréquent de tous les points de données de la même classe. Il est même préférable de substituer la distribution de probabilité de toutes les valeurs d'attribut à la valeur d'attribut manquante et de diviser l'exemple d'apprentissage en branches en fonction de cette distribution. C'est d'ailleurs une des raisons de l'apparition de valeurs non entières dans les expressions entre parenthèses à côté des nœuds feuilles de l'arbre C4.5.

Les valeurs manquantes peuvent se produire non seulement lors de l'apprentissage, mais également lors de la classification. Celles-ci sont traitées de la même manière que lors de l'apprentissage.

8 Il serait préférable d'utiliser directement l'erreur sur les données de test. Au moins lorsque la quantité de données d'apprentissage est suffisante pour justifier un ensemble de tests distinct.

8.4.9 Résumé

L'apprentissage des arbres de décision est une approche privilégiée pour les tâches de classification. Les raisons en sont sa simplicité d'application et sa rapidité. Sur un ensemble de données d'environ 10 000 points de données LEXMED avec 15 attributs chacun, C4.5 nécessite environ 0,3 seconde pour l'apprentissage. C'est très rapide par rapport aux autres algorithmes d'apprentissage.

Pour l'utilisateur, il est également important, cependant, que l'arbre de décision en tant que modèle appris puisse être compris et éventuellement modifié. Il n'est pas non plus difficile de transformer automatiquement un arbre de décision en une série d'instructions if-then-else et ainsi de l'intégrer efficacement dans un programme existant.

Parce qu'un algorithme glouton est utilisé pour la construction de l'arbre ainsi que pendant l'élagage, les arbres sont en général sous-optimaux. L'arbre de décision découvert a généralement un taux d'erreur relativement faible. Cependant, il existe potentiellement un meilleur arbre, car la recherche gloutonne heuristique de C4.5 préfère les petits arbres et les attributs avec un gain d'information élevé au sommet de l'arbre. Pour les attributs à plusieurs valeurs, la formule présentée pour le gain d'information présente des faiblesses. Des alternatives à cela sont données dans [Mit97].

8.5 Apprentissage des réseaux bayésiens

Insecte. 7.4, il a été montré comment construire un réseau bayésien manuellement. Nous allons maintenant introduire des algorithmes pour l'induction de réseaux bayésiens. Semblable au processus d'apprentissage décrit précédemment, un réseau bayésien est automatiquement généré à partir d'un fichier contenant des données d'apprentissage. Ce processus est généralement décomposé en deux parties.

1. Apprentissage de la structure du réseau : Pour des variables données, la topologie du réseau est générée à partir des données d'apprentissage. Cette première étape est de loin la plus difficile et fera l'objet d'une attention particulière plus tard.
2. Apprentissage des probabilités conditionnelles : Pour les topologies de réseau connues, les CPT doivent être renseignés avec des valeurs. Si suffisamment de données d'entraînement sont disponibles, toutes les probabilités conditionnelles nécessaires peuvent être estimées en comptant les fréquences dans les données. Cette étape peut être automatisée relativement facilement.

Nous allons maintenant expliquer comment les réseaux bayésiens apprennent en utilisant un algorithme simple de [Jen01].

8.5.1 Apprentissage de la structure du réseau

Lors du développement d'un réseau bayésien (voir section 7.4.6), la dépendance causale des variables doit être prise en compte afin d'obtenir un travail de réseau simple de bonne qualité. Le développeur humain s'appuie sur des connaissances de base, qui ne sont pas disponibles pour la machine. Par conséquent, cette procédure ne peut pas être facilement automatisée.

Trouver une structure optimale pour un réseau bayésien peut être formulé comme un problème de recherche classique. Soit un ensemble de variables V1,...,Vn et un fichier avec des données d'apprentissage. On cherche un ensemble d'arêtes dirigées sans cycles entre les nœuds

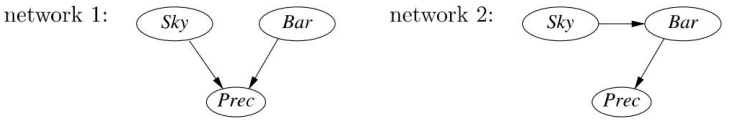


Fig. 8.28 Deux réseaux bayésiens pour modéliser l'exemple de prévision météorologique de l'exercice 7.3 à la page 158

V1,...,Vn, c'est-à-dire un graphe orienté acyclique (DAG) qui reproduit au mieux les données sous-jacentes.

On observe d'abord l'espace de recherche. Le nombre de DAG différents croît plus qu'exponentiellement avec le nombre de nœuds. Pour cinq nœuds, il y a 29281 et pour neuf nœuds environ 1015 DAG différents [MDBM00]. Ainsi, une recherche combinatoire non informée (voir section 6.2) dans l'espace de tous les graphes avec un ensemble donné de variables est sans espoir si le nombre de variables augmente. Des algorithmes heuristiques doivent donc être utilisés. Cela pose la question d'une fonction d'évaluation pour les réseaux bayésiens. Il est possible de mesurer l'erreur de classification d'un réseau lors de l'application à un jeu de données de test, comme cela se fait par exemple en C4.5 (voir § 8.4). Pour cela, cependant, les probabilités calculées par le réseau bayésien doivent être mappées à une décision.

Une mesure directe de la qualité d'un réseau peut être prise sur la distribution de probabilité. Nous supposons qu'avant la construction du réseau à partir des données, nous pourrions déterminer (estimer) la distribution. Ensuite, nous commençons la recherche dans l'espace de tous les DAG, estimons la valeur des CPT pour chaque DAG (c'est-à-dire pour chaque réseau bayésien) à l'aide des données, et à partir de là, nous calculons la distribution et la comparons à la distribution connue de les données. Pour la comparaison des distributions, nous aurons évidemment besoin d'une métrique de distance.

Considérons l'exemple de prévision météorologique de l'exercice 7.3 à la page 158 avec les trois variables Sky, Bar, Prec et la distribution

$$P(\text{Ciel,Bar,Préc}) = (0.40, 0.07, 0.08, 0.10, 0.09, 0.11, 0.03, 0.12).$$

Sur la Fig. 8.28 , deux réseaux bayésiens sont présentés, que nous allons maintenant comparer en fonction de leur qualité. Chacun de ces réseaux fait une hypothèse d'indépendance, qui est validée en ce sens que nous déterminons la distribution du réseau et que nous la comparons ensuite avec la distribution d'origine (voir l'exercice 8.15 à la page 219).

Puisque, pour des variables prédéterminées constantes, la distribution est clairement représentée par un vecteur de longueur constante, nous pouvons calculer la norme euclidienne de la différence des deux vecteurs comme une distance entre les distributions. Nous définissons

$$dq(x, y) = \sum (x_i - y_i)^2$$

comme la somme des carrés des distances des composantes vectorielles et calculer la distance dq (Pa,P) = 0,0029 de la distribution Pa du réseau 1 à la distribution d'origine. Pour le réseau 2, nous calculons dq (Pb,P) = 0,014. Il est clair que le réseau 1 est une meilleure approximation de la distribution. Souvent, au lieu de la distance au carré, la

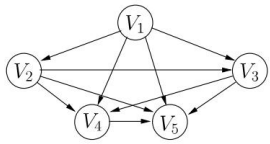


Fig. 8.29 Le réseau maximal à cinq variables et les arêtes (Vi, Vj) qui remplissent la condition i<j

distance dite de Kullback – Leibler

$$dk(x, y) = \sum yi(\log 2\, yi - \log 2\, xi),$$

une métrique de la théorie de l'information, est utilisée. Avec cela, nous calculons dk(Pa,P) = 0,017 et dk(Pb,P) = 0,09 et arrivons à la même conclusion qu'auparavant. Il faut s'attendre à ce que les réseaux avec de nombreuses arêtes se rapprochent mieux de la distribution que ceux avec peu d'arêtes. Si toutes les arêtes du réseau sont construites, cela devient alors très confus et crée un risque de surajustement, comme c'est le cas dans de nombreux autres algorithmes d'apprentissage. Pour éviter le surajustement, nous donnons aux petits réseaux un poids plus important en utilisant une fonction d'évaluation heuristique

$$f(N) = \text{Taille}(N) + w \cdot dk(P_N, P).$$

Ici, Size(N) est le nombre d'entrées dans les CPT et P N est la distribution du réseau N. w est un facteur de pondération, qui doit être ajusté manuellement.

L'algorithme d'apprentissage des réseaux bayésiens calcule ainsi l'évaluation heuristique f(N) pour de nombreux réseaux différents puis choisit le réseau avec la plus petite valeur. Comme mentionné précédemment, la difficulté consiste en la réduction de l'espace de recherche pour la topologie de réseau que nous recherchons. Comme algorithme simple, il est possible, à partir d'un ordre (par exemple causal) des variables V1,...,Vn, de n'inclure dans le graphe que les arêtes pour lesquelles i<j . Commençons par le modèle maximal qui remplit cette condition. Ce réseau est représenté sur la Fig. 8.29 pour cinq variables ordonnées.

Maintenant, par exemple dans l'esprit d'une recherche gloutonne (cf. § 6.3.1), une arête après l'autre est supprimée jusqu'à ce que la valeur f ne diminue plus.

Cet algorithme n'est pas pratique pour les grands réseaux sous cette forme. Le grand espace de recherche, le réglage manuel du poids w et la comparaison nécessaire avec une distribution d'objectifs P en sont les raisons, car ceux-ci peuvent simplement devenir trop grands ou l'ensemble de données disponible peut être trop petit.

En fait, les recherches sur l'apprentissage des réseaux bayésiens battent toujours leur plein, et il existe un grand nombre d'algorithmes proposés, par exemple l'algorithme EM (voir la section 8.7.2), les méthodes de Monte Carlo par chaîne de Markov et l'échantillonnage de Gibbs [DHS01, Jor99, Jen01, HTF09]. Outre l'apprentissage par lots, qui a été présenté ici, dans lequel le réseau est généré une fois à partir de l'ensemble de données, il existe également des algorithmes incrémentaux, dans lesquels chaque nouveau cas individuel est utilisé pour améliorer le réseau. Des implémentations de ces algorithmes existent également, comme Hugin (www.hugin.com) et Bayesware (www.bayesware.com).

8.6 Le classificateur naïf de Bayes

Dans la Fig. 7.14 à la page 152, le diagnostic d'appendicite a été modélisé comme un réseau bayésien. Étant donné que les arêtes dirigées commencent à un nœud de diagnostic et qu'aucune ne s'y termine, la formule de Bayes doit être utilisée pour répondre à une requête de diagnostic. Pour les symptômes S1,...,Sn et le diagnostic de valeur k D avec les valeurs b1,...,bk, nous calculons la probabilité

$$P(D|S1,...,Sn) = P \frac{P(S1,...,Sn|D) \cdot P(D)}{(S1,...,Sn)},$$

pour le diagnostic compte tenu des symptômes du patient. Dans le pire des cas, c'est-à-dire s'il n'y avait pas de variables indépendantes, toutes les combinaisons de tous les symptômes et D devraient être déterminées pour toutes les 20 643 840 probabilités de la distribution P(S1,...,Sn,D). Cela nécessiterait une énorme base de données. Dans le cas du réseau bayésien de LEX MED, le nombre de valeurs nécessaires (dans les CPT) est réduit à 521. Le réseau peut cependant être encore simplifié en supposant que toutes les variables de symptôme sont conditionnellement indépendantes étant donné D, c'est-à-dire :

$$P(S1,...,Sn|D) = P(S1|D) \cdots P(Sn|D).$$

Le réseau bayésien pour l'appendicite est ensuite simplifié à l'étoile illustrée à la Fig. 8.30 à la page 203.
Ainsi on obtient la formule

$$S1,...,Sn) = \frac{P(D)n\ P\ (Si|D)\ i=1\ P}{P(S1,...,Sn)}. \qquad (8.8)$$

Les probabilités calculées sont transformées en une décision par un simple classificateur Bayes naïf, qui choisit le maximum P (D = di |S1,...,Sn) parmi toutes les valeurs di dans D. C'est-à-dire qu'il détermine

$$dNaive\text{-}Bayes = argmaxi \quad \{1,...,k\}\ P\ (D = di\ |S1,...,Sn).$$

Comme le dénominateur dans (8.8) est constant, il peut être omis lors de la maximisation, ce qui donne la formule naïve de Bayes

$$dNaive\text{-}Bayes = argmaxi \quad \{1,...,k\}\ P\ (D = di) \prod_{j=1}^{n} P\ (Sj\ |D).$$

Parce que plusieurs nœuds ont maintenant moins d'ancêtres, le nombre de valeurs nécessaires pour décrire la distribution LEXMED dans les CPT diminue, selon (7.22) page 151, à

$$6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 4 + 3 \cdot 4 + 10 \cdot (1 \cdot 4) + 1 = 141.$$

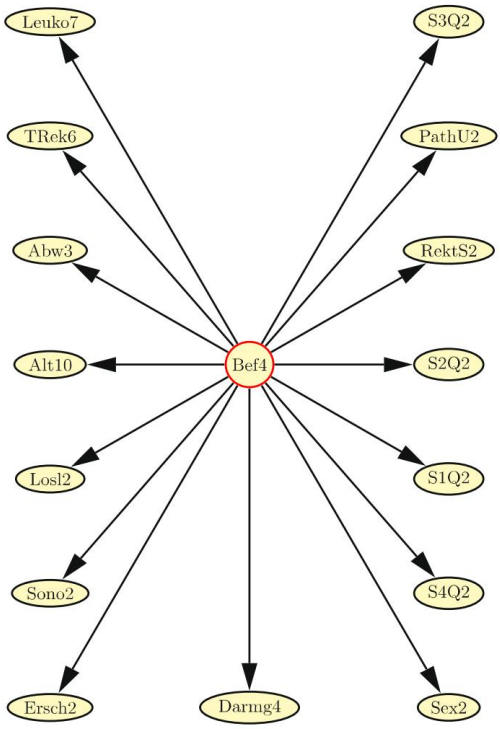


Fig. 8.30 Réseau bayésien pour l' application LEXMED avec l'hypothèse que tous les symptômes sont conditionnellement indépendants compte tenu du diagnostic

Pour un système de diagnostic médical comme LEXMED, cette simplification ne serait pas acceptable. Mais pour des tâches comportant de nombreuses variables indépendantes, Bayes naïf est partiellement voire très bien adapté, comme nous le verrons dans l'exemple de classification de texte.

Soit dit en passant, la classification naïve de Bayes est tout aussi expressive que le système de score linéaire décrit dans la Sect. 7.3.1 (voir Exercice 8.16 à la page 219). Autrement dit, tous les scores partagent l'hypothèse sous-jacente selon laquelle tous les symptômes sont conditionnellement indépendants compte tenu du diagnostic. Néanmoins, les scores sont encore utilisés en médecine aujourd'hui. Bien qu'il ait été généré à partir d'une meilleure base de données représentative, le score d'Ohmann comparé à LEXMED dans la Fig. 7.10 à la page 142 a une moins bonne qualité de diagnostic. Son expressivité limitée en est certainement une raison. Par exemple, comme avec naïf Bayes, il n'est pas possible de modéliser les dépendances entre les symptômes en util scores.

Estimation des probabilités Si nous observons la formule naïve de Bayes dans (8.8) à la page 202, nous voyons que l'expression entière devient nulle dès que l'un des facteurs P (Si |D) du côté droit devient nul. Théoriquement, il n'y a rien de mal ici. En pratique, cependant, cela peut conduire à des effets très inconfortables si les P (Si | D) sont petits, car ceux-ci sont estimés en comptant les fréquences et en les substituant

dans

$$P(Si = x|D = y) = \frac{|Si = x \quad D = y|}{|D = y|}.$$

Supposons que pour les variables Si : P (Si = x|D = y) = 0,01 et qu'il y ait 40 cas d'apprentissage avec D = y. Alors avec une probabilité élevée, il n'y a pas de cas d'apprentissage avec Si = x et D = y, et nous estimons P (Si = x|D = y) = 0. Pour une valeur différente D = z, supposons que les relations sont situées de manière similaire, mais l'estimation donne des valeurs supérieures à zéro pour tout P (Si = x|D = z). Ainsi, la valeur D = z est toujours préférée, ce qui ne reflète pas la distribution de probabilité réelle. Par conséquent, lors de l'estimation des probabilités, la formule

$$P(A|B) \approx \frac{|A \quad B|}{|B|} = \frac{nAB}{nB}$$

est remplacé par

$$P(A|B) \approx \frac{nAB + mp}{nB + m},$$

où p = P (A) est la probabilité a priori pour A, et m est une constante qui peut être choisie librement et est connue sous le nom de "taille de données équivalente". Plus m devient grand, plus le poids de la probabilité a priori est grand par rapport à la valeur déterminée à partir de la fréquence mesurée.

8.6.1 Classification de texte avec Naive Bayes

Naive Bayes est aujourd'hui très performant et prolifique dans la classification de textes. Son application principale, et en même temps très importante, est le filtrage automatique des e-mails en e-mails souhaités et indésirables, ou spams. Dans les filtres anti-spam tels que SpamAssassin [Sch04], entre autres méthodes, un classificateur Bayes naïf est utilisé pour apprendre à séparer les e-mails souhaités des spams. SpamAssassin est un système hybride qui effectue un premier filtrage à l'aide de listes noires et blanches. Les listes noires sont des listes d'adresses e-mail bloquées des expéditeurs de spam dont les e-mails sont toujours supprimés, et les listes blanches sont celles des expéditeurs dont les e-mails sont toujours livrés. Après ce préfiltrage, les emails restants sont classés par le classificateur naïf Bayes selon leur contenu réel, c'est-à-dire selon le texte. La valeur de classe détectée est ensuite évaluée par un score, ainsi que d'autres attributs de l'en-tête de l'e-mail tels que le domaine de l'expéditeur, le type MIME, etc., puis enfin filtrée.

Ici, la capacité d'apprentissage du filtre Bayes naïf est très importante. Pour cela, l'utilisateur doit d'abord classer manuellement un grand nombre d'e-mails comme souhaité ou spam. Ensuite, le filtre est formé. Pour rester à jour, le filtre doit être régulièrement recyclé. Pour cela, l'utilisateur doit classer correctement tous les e-mails qui ont été faussement classés par le filtre, c'est-à-dire les mettre dans les dossiers appropriés. Le filtre est alors continuellement recyclé avec ces e-mails.

Outre le filtrage anti-spam, il existe de nombreuses autres applications de classification automatique des textes. Parmi les applications importantes, citons le filtrage des entrées indésirables dans les forums de discussion sur Internet et le suivi des sites Web au contenu criminel tels que les activités militantes ou terroristes, la pédopornographie ou le racisme. Il peut également être utilisé pour personnaliser les moteurs de recherche en fonction des préférences de l'utilisateur afin de mieux classer les résultats de la recherche. Dans le cadre industriel et scientifique, la recherche à l'échelle de l'entreprise dans les bases de données ou dans la littérature est au premier plan de la recherche. Grâce à sa capacité d'apprentissage, un filtre peut s'adapter aux habitudes et aux souhaits de chaque utilisateur.

Nous présenterons l'application de Bayes naïf à l'analyse de texte sur un court ex ample texte d'Alan Turing de [Tur50] :

> « On peut espérer que les machines finiront par concurrencer les hommes dans tous les domaines purement intellectuels. Mais quels sont les meilleurs pour commencer ? C'est même une décision difficile. Beaucoup de gens pensent qu'une activité très abstraite, comme jouer aux échecs, serait la meilleure. On peut également soutenir qu'il est préférable de doter la machine des meilleurs organes sensoriels que l'argent puisse acheter, puis de lui apprendre à comprendre et à parler anglais. Ce processus pourrait suivre l'enseignement normal d'un enfant. Les choses seraient signalées et nommées, etc. Encore une fois, je ne sais pas quelle est la bonne réponse, mais je pense que les deux approches devraient être essayées.

Supposons que des textes comme celui-ci soient divisés en deux classes : « je » pour intéressant et « ¬je pour inintéressant ». Supposons également qu'il existe une base de données de textes déjà classés. Quels attributs utiliser ? Dans une approche classique de la construction d'un réseau bayésien, on définit un ensemble d'attributs tels que la longueur du texte, la longueur moyenne de la phrase, la fréquence relative de signes de ponctuation spécifiques, la fréquence de plusieurs mots importants tels que "je" , « machines », etc. Lors d'une classification à l'aide de Bayes naïf, en revanche, un algorithme étonnamment primitif est sélectionné. Pour chacune des n positions de mots dans le texte, un attribut si est défini. Tous les mots apparaissant dans le texte sont autorisés comme valeurs possibles pour toutes les positions si . Maintenant pour les classes I et ¬I les valeurs

$$P(je|s1,...,sn) = c \cdot P(je) \prod_{je=1}^{n} P(si|I) \tag{8.9}$$

et P (¬I |s1,...,sn) doivent être calculés puis la classe avec la valeur maximale sélectionnée. Dans l'exemple ci-dessus avec un total de 113 mots, cela donne

$$P(I|s1,...,sn) = c \cdot P(je) \cdot P(s1 = \text{"nous"}|je) \cdot P(s2 = \text{"peut"}|je) \cdots P(s113 = \text{"devrait"}|je)$$

et

$$P(\neg I|s1,...,sn) = c \cdot P(\neg I) \cdot P(s1 = \text{"Nous"}|\neg I) \cdot P(s2 = \text{"peut"}|\neg I) \cdots P(s113 = \text{"devrait"}|\neg I).$$

L'apprentissage ici est assez simple. Les probabilités conditionnelles P (si |I), P (si |¬I) et les probabilités a priori P (I), P (¬I) doivent simplement être calculées. Nous supposons maintenant en plus que les P (si |I) ne dépendent pas de la position dans le texte. Ce

signifie, par exemple, que

$$P(s61 = \text{"et"} | je) = P(s69 = \text{"et"} | je) = P(s86 = \text{"et"} | je).$$

On pourrait ainsi utiliser l'expression P (et|I), avec la nouvelle variable binaire et, comme probabilité d'occurrence de « et » à une position arbitraire.

La mise en œuvre peut être quelque peu accélérée si l'on trouve la fréquence ni de chaque mot wi qui apparaît dans le texte et utilisez la formule

$$P(je|s1,...,sn) = c \cdot P(je) \prod_{je=1}^{je*} P(wi|I)^{ni} \tag{8.10}$$

qui équivaut à (8.9) à la page 205. Veuillez noter que l'index i dans le produit ne va qu'au nombre I de mots différents qui apparaissent dans le texte.

Malgré sa simplicité, naïve Bayes fournit d'excellents résultats pour la classification de texte. Les filtres anti-spam qui fonctionnent avec des Bayes naïfs atteignent des taux d'erreur bien inférieurs à un pour cent. Les systèmes DSPAM et CRM114 peuvent même être si bien entraînés qu'ils ne classent respectivement qu'un seul e-mail sur 7000 ou 8000. Cela correspond à une exactitude de près de 99,99 %.

8.7 Regroupement

Si nous recherchons dans un moteur de recherche le terme "mars", nous obtiendrons des résultats comme "la planète mars" et "Conglomérat de chocolat, confiserie et boissons" qui sont sémantiquement assez différents. Dans l'ensemble des documents découverts, il y a deux groupes sensiblement différents . Google, par exemple, répertorie toujours les résultats de manière non structurée. Il serait préférable que le moteur de recherche sépare les clusters et les présente à l'utilisateur en conséquence, car l'utilisateur s'intéresse généralement à un seul des clusters.

La distinction du regroupement par rapport à l'apprentissage supervisé est que les données de formation ne sont pas étiquetées. Ainsi, la pré-structuration des données par le superviseur fait défaut. Au contraire, trouver des structures est tout l'intérêt du regroupement. Dans l'espace des données d'apprentissage, on trouve des accumulations de données telles que celles de la Fig. 8.31 à la page 207 . Dans un cluster, la distance des points voisins est généralement inférieure à la distance entre les points des différents clusters. Par conséquent, le choix d'une métrique de distance appropriée pour les points, c'est-à-dire pour les objets à grouper et pour les clusters, est d'une importance fondamentale. Comme précédemment, nous supposons dans ce qui suit que chaque objet de données est décrit par un vecteur d'attributs numériques.

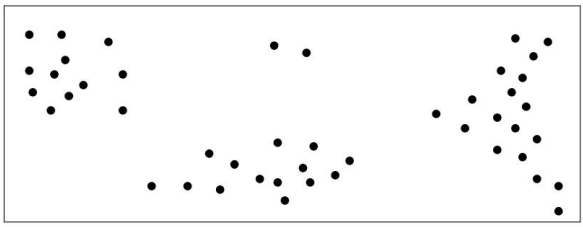


Fig. 8.31 Exemple bidimensionnel simple avec quatre clusters clairement séparés

8.7.1 Mesures de distance

Ainsi pour chaque application, les différentes métriques de distance sont définies pour la distance d entre deux vecteurs x et y dans Rn . La plus courante est la distance euclidienne

$$de(x, y) = \sqrt{\sum_{je=1}^{n} (xi - yi)\,2.}$$

Un peu plus simple est la somme des distances au carré

$$dq(x, y) = \sum_{je=1}^{n} (xi - yi)^{2},$$

qui, pour les algorithmes dans lesquels seules les distances sont comparées, est équivalente à la distance euclidienne (Exercice 8.19 page 220). La distance de Manhattan susmentionnée est également utilisée

$$dm(x, y) = \sum_{je=1}^{n} |xi - yi|$$

ainsi que la distance de la composante maximale

$$d\infty(x, y) = \max_{i=1,\ldots,n} |xi - yi|$$

qui est basé sur la norme maximale. Lors de la classification de texte, la projection normalisée des deux vecteurs l'un sur l'autre, c'est-à-dire le produit scalaire normalisé

$$\frac{xy|}{x||y|}$$

est fréquemment calculé, où |x| est la norme euclidienne de x. Parce que cette formule est une métrique pour la similarité des deux vecteurs, en tant que métrique de distance l'inverse

$$ds(x, y) = \frac{|x||y|}{xy}$$

peut être utilisé, ou ">" et "<" peuvent être permutés pour toutes les comparaisons. Dans la recherche d'un texte, les attributs x1,...,xn sont calculés de manière similaire à Bayes naïf en tant que composantes du vecteur x comme suit. Pour un dictionnaire de 50 000 mots, la valeur xi

est égal à la fréquence du ième mot du dictionnaire dans le texte. Puisque normalement presque tous les composants sont nuls dans un tel vecteur, lors du calcul du produit scalaire, presque tous les termes de la sommation sont nuls. En exploitant ce type d'informations, la mise en œuvre peut être considérablement accélérée (Exercice 8.20 à la page 220).

8.7.2 k-Means et l'algorithme EM

Chaque fois que le nombre de clusters est déjà connu à l'avance, l' algorithme k-means peut être utilisé. Comme son nom l'indique, les clusters k sont définis par leur valeur moyenne. Tout d'abord, les k points médians du cluster μ1 ,...,μk sont initialisés aléatoirement ou manuellement. Ensuite, les deux étapes suivantes sont répétées : • Classification de toutes les données jusqu'à leur point médian de cluster le plus proche • Recalcul du point médian de cluster.

Le schéma suivant se traduit par un algorithme :

```
K-MEANS(x1,..., xn,k) initialise
μ1 ,...,μk (par exemple aléatoirement)
Répéter
    classer x1,..., xn au μi le plus proche de chacun
    recalculer μ1 ,...,μk Jusqu'à
ce qu'il n'y ait aucun changement dans
μ1 ,...,μk Retour(μ1 ,...,μk )
```

Le calcul du milieu de cluster μ pour les points x1,..., xl est fait par

$$\mu = \frac{1}{je} \sum_{je=1}^{je} xi .$$

L'exécution sur un exemple est illustrée à la Fig. 8.32 à la page 209 pour le cas de deux classes. On voit comment après trois itérations, les centres de classes, qui ont d'abord été choisis au hasard, se stabilisent. Bien que cet algorithme ne garantisse pas la convergence, il converge généralement très rapidement. Cela signifie que le nombre d'étapes d'itération est généralement beaucoup plus petit que le nombre de points de données. Sa complexité est O(ndkt), où n est le nombre total de points, d la dimensionnalité de l'espace des caractéristiques et t le nombre d'étapes d'itération.

Dans de nombreux cas, la nécessité de donner le nombre de classes à l'avance pose une limitation gênante. C'est pourquoi nous introduirons ensuite un algorithme plus flexible.

Avant cela, cependant, nous mentionnerons l' algorithme EM, qui est une variante continue des k-moyennes, car il ne fait pas une affectation ferme des données aux classes, mais renvoie pour chaque point la probabilité qu'il appartienne à la divers cours.

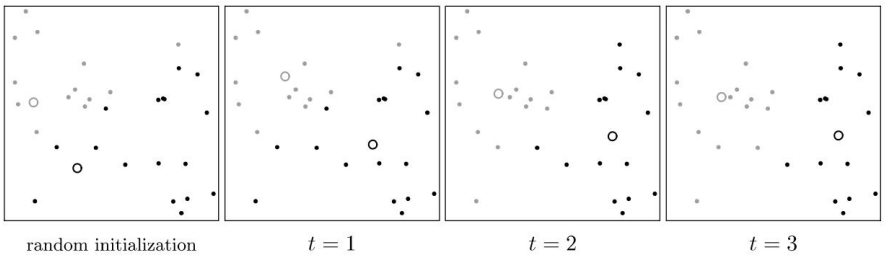


Fig. 8.32 k-means avec deux classes (k = 2) appliquées à 30 points de données. Tout à gauche se trouve l'ensemble de données avec les centres initiaux, et à droite se trouve le cluster après chaque itération. Après trois itérations, la convergence est atteinte

Ici, nous devons supposer que le type de distribution de probabilité est connu. La distribution normale est souvent utilisée. La tâche de l'algorithme EM est de déterminer les paramètres (moyenne μi et matrices de covariance Σi des k distributions normales multidimensionnelles) pour chaque cluster. Comme pour les k-moyennes, les deux étapes suivantes sont exécutées de manière répétée :

Espérance : Pour chaque point de données, la probabilité P (Cj |xi) qu'il appartienne à chaque cluster est calculée.

Maximisation : En utilisant les probabilités nouvellement calculées, les paramètres de la distribution sont recalculés.

Ainsi, un regroupement plus doux est obtenu, ce qui, dans de nombreux cas, conduit à de meilleurs résultats. Cette alternance entre espérance et maximisation donne son nom à l'algorithme. En plus du clustering, par exemple, l'algorithme EM est utilisé pour apprendre les réseaux bayésiens [DHS01].

8.7.3 Regroupement hiérarchique

Dans le clustering hiérarchique, nous commençons avec n clusters constitués d'un point chacun. Ensuite, les clusters voisins les plus proches sont combinés jusqu'à ce que tous les points aient été combinés en un seul cluster, ou jusqu'à ce qu'un critère de terminaison soit atteint. On obtient le schéma

```
HIERARCHICALCLUSTERING(x1,..., xn,k) initialiser
C1 = {x1},...,Cn = {xn}
Répéter
    Trouver deux clusters Ci et Cj avec la plus petite distance
    Combiner Ci et Cj
Jusqu'à ce que la condition de terminaison
soit atteinte Retour(arbre avec clusters)
```

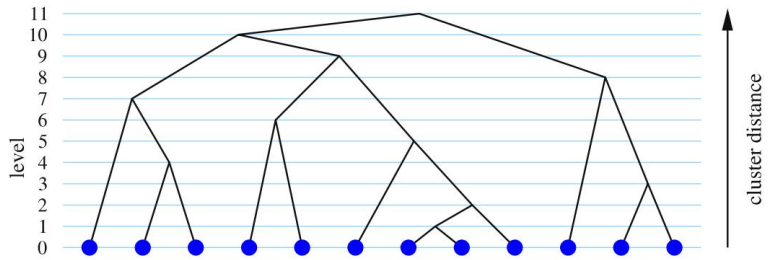


Fig. 8.33 Dans le clustering hiérarchique, les deux clusters avec la plus petite distance sont combinés à chaque étape

La condition de terminaison pourrait être choisie comme, par exemple, un nombre souhaité de grappes ou une distance maximale entre les grappes. Sur la figure 8.33 , cet algorithme est représenté schématiquement sous la forme d'un arbre binaire, dans lequel de bas en haut à chaque étape, c'est-à-dire à chaque niveau, deux sous-arbres sont connectés. Au niveau supérieur, tous les points sont unifiés en un seul grand cluster.

On ne sait pas encore comment les distances entre les clusters sont calculées. En effet, dans la section précédente nous avons défini diverses métriques de distance pour les points, mais celles-ci ne peuvent pas être utilisées sur les clusters. Une métrique pratique et souvent utilisée est la distance entre les deux points les plus proches dans les deux clusters Ci et Cj :

$$d_{min}(C_i, C_j) = \min d(x, y). \; x \in C_i, \; y \in C_j$$

Nous obtenons ainsi l' algorithme du plus proche voisin, dont l'application est illustrée à la Fig. 8.34 page 211. L'exemple montre en [9] tree.10 On voit que cet algorithme génère un enjambement minimum outre que les deux algorithmes décrits génèrent des clusters assez différents. Cela nous indique que pour les graphes avec des clusters qui ne sont pas clairement séparés, le résultat dépend fortement de l'algorithme ou de la métrique de distance choisie.

Pour une implémentation efficace de cet algorithme, nous créons d'abord une matrice d'adjacence dans laquelle les distances entre tous les points sont enregistrées, ce qui nécessite O(n2) temps et mémoire. Si le nombre de clusters n'a pas de limite supérieure, la boucle va itérer n − 1 fois et le temps de calcul asymptotique devient O(n3).

Pour calculer la distance entre deux clusters, on peut aussi utiliser la distance be
entre les deux points les plus éloignés

$$d_{max}(C_i, C_j) = \max d(x, y) \; x \in C_i, y \in C_j$$

9L'algorithme du plus proche voisin ne doit pas être confondu avec la méthode du plus proche voisin pour la classe sification de la Sect. 8.3.

10Un arbre couvrant minimum est un graphe acyclique non orienté avec la somme minimale des longueurs d'arêtes.

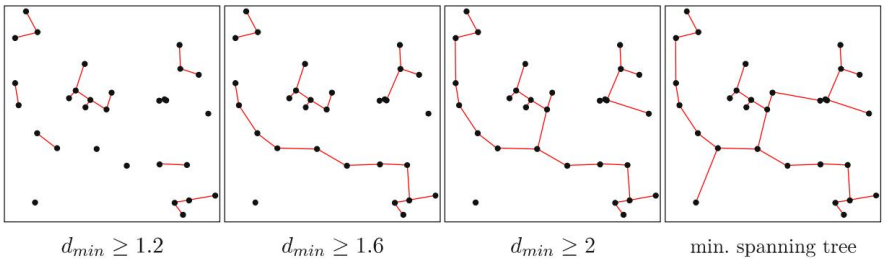


Fig. 8.34 L'algorithme du plus proche voisin appliqué aux données de la Fig. 8.32 à la page 209 à différents niveaux avec 12, 6, 3, 1 clusters

et obtenir l' algorithme du voisin le plus éloigné. Alternativement, la distance du milieu du cluster dμ(Ci, Cj) = d(μi,μj) est utilisée. Outre l'algorithme de clustering
présentés ici, il en existe bien d'autres, pour lesquels nous renvoyons le lecteur à [DHS01]
pour une étude plus approfondie.

8.8 L'exploration de données en pratique

Tous les algorithmes d'apprentissage présentés jusqu'à présent peuvent être utilisés comme outils d'exploration de données. Pour l'utilisateur, il est cependant parfois assez difficile de s'habituer à un nouveau logiciel
outils pour chaque application et de mettre les données à analyser dans le
format approprié pour chaque cas particulier.

Un certain nombre de systèmes d'exploration de données résolvent ces problèmes. La plupart de ces systèmes offrent une interface utilisateur graphique pratique avec divers outils de visualisation de
données, pour le prétraitement, comme la manipulation des valeurs manquantes, et pour l'analyse. Pour analyse, les algorithmes d'apprentissage présentés ici sont utilisés, entre autres.

La bibliothèque Java open-source complète WEKA mérite une mention spéciale.
Il offre un grand nombre d'algorithmes et simplifie le développement de nouveaux algorithmes.

Le système librement disponible KNIME, que nous présenterons brièvement dans le
section suivante, offre une interface utilisateur pratique et tous les types d'outils
mentionné ci-dessus. KNIME utilise également les modules WEKA. De plus il offre une
un moyen simple de contrôler le flux de données des outils de visualisation, de prétraitement et d'analyse choisis avec un éditeur graphique. Un grand nombre d'autres systèmes offrent entre-temps des fonctionnalités assez similaires, comme le projet open-source RapidMiner(www.rapidminer.com), le système Clementine (www.spss.com/clémentine) vendu par SPSS, et le cadre analytique KXEN (www.kxen.com).

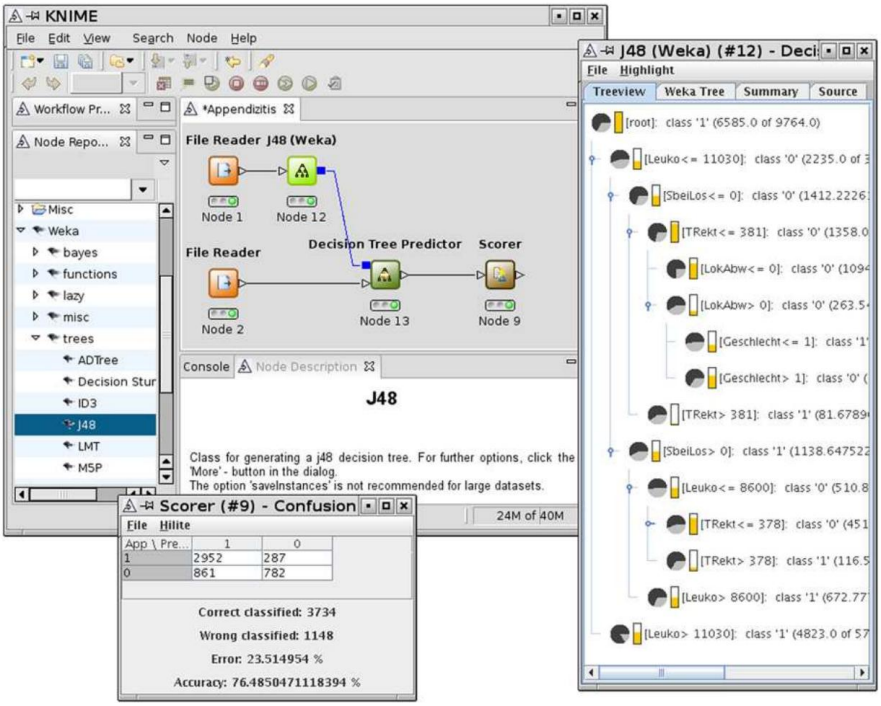


Fig. 8.35 L'interface utilisateur KNIME avec deux vues supplémentaires, qui montrent l'arbre de décision et la matrice de confusion

8.8.1 L'outil d'exploration de données KNIME

En utilisant les données LEXMED , nous allons maintenant montrer comment extraire des connaissances à partir de données à l'aide de KNIME (Konstanz Information Miner, www.knime.org). Nous générons d'abord un arbre de décision comme le montre la Fig. 8.35. Après avoir créé un nouveau projet, un workflow est construit graphiquement. Pour ce faire, les outils appropriés sont simplement retirés du référentiel de nœuds avec la souris et glissés dans la fenêtre principale du workflow.

Les données d'entraînement et de test du fichier C4.5 peuvent être lues sans problème avec les deux nœuds de lecture de fichier. Cependant, ces nœuds peuvent également être configurés assez facilement pour d'autres formats de fichiers. Le feu de signalisation latéral sous le nœud indique son état (pas prêt, configuré, exécuté). Ensuite, le nœud J48 est sélectionné dans la bibliothèque WEKA [WF01], qui contient une implémentation Java de C4.5. La configuration pour cela est assez simple. Maintenant, un nœud prédicteur est choisi, qui applique l'arbre généré aux données de test. Il insère une nouvelle colonne dans la table de données de test "Prédiction" avec la classification générée par l'arbre. À partir de là, le nœud de score calcule la matrice de confusion illustrée sur la figure, qui donne le nombre de cas correctement classés pour les deux classes dans la diagonale, ainsi que le nombre de points de données faux positifs et faux négatifs.

Tableau 8.1 Données de formation pour l'agent de tri des pommes

Taille [cm]	8	8 6 3. . .
Couleur	0,1	0,3 0,9 0,8 . . .
Classe de marchandises	B	A A B . . .

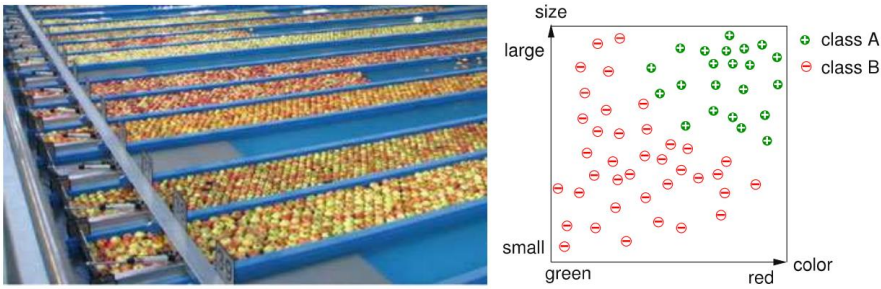
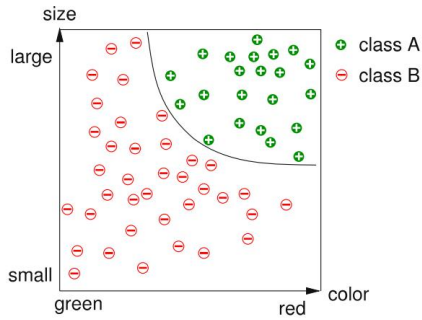


Fig. 8.2 Équipement de tri de pommes de la société BayWa à Kressbronn et quelques pommes classées dans les classes de marchandises A et B dans l'espace caractéristique (Photo : BayWa)

Fig. 8.3 La courbe tracée dans le diagramme divise les classes et peut ensuite être appliquée à de nouvelles pommes arbitraires



Une visualisation des données est répertoriée sous forme de points dans un diagramme de dispersion à droite de la Fig. 8.2.

La tâche en apprentissage automatique consiste à générer une fonction à partir des données classifiées collectées qui calcule la valeur de classe (A ou B) pour une nouvelle pomme à partir des deux caractéristiques taille et couleur. Dans la Fig. 8.3, une telle fonction est représentée par la ligne de division tracée à travers le diagramme. Toutes les pommes avec un vecteur de caractéristiques en bas à gauche de la ligne sont placées dans la classe B, et toutes les autres dans la classe A.

Dans cet exemple, il est encore très simple de trouver une telle ligne de séparation pour les deux classes. C'est évidemment une tâche plus difficile, et surtout beaucoup moins visualisable, lorsque les objets à classer sont décrits non seulement par deux, mais par de nombreux traits. En pratique, 30 fonctionnalités ou plus sont généralement utilisées. Pour n traits, la tâche consiste à trouver un hyperplan à n − 1 dimensions dans l' espace des traits à n dimensions qui divise au mieux les classes. Une « bonne » division signifie que le pourcentage d'objets faussement classés est le plus faible possible.

Fig. 8.4 Structure fonctionnelle d'un agent d'apprentissage pour le tri des pommes (à gauche) et en général (à droite)

Un classificateur mappe un vecteur de caractéristiques à une valeur de classe. Ici, il a un nombre fixe, généralement petit, d'alternatives. Le mappage souhaité est également appelé fonction cible. Si la fonction cible ne correspond pas à un domaine fini, il ne s'agit pas d'une classification, mais plutôt d'un problème d'approximation. Déterminer la valeur marchande d'un stock à partir de caractéristiques données est un tel problème d'approximation. Dans les sections suivantes, nous présenterons plusieurs agents d'apprentissage pour les deux types de mappages.

L'agent d'apprentissage Nous pouvons formellement décrire un agent d'apprentissage comme une fonction qui fait correspondre un vecteur de caractéristiques à une valeur de classe discrète ou en général à un nombre réel. Cette fonction n'est pas programmée, mais elle apparaît ou se modifie pendant la phase d'apprentissage, influencée par les données d'apprentissage. Dans la Fig. 8.4 , un tel agent est présenté dans l'exemple du tri des pommes. Pendant l'apprentissage, l'agent est alimenté avec les données déjà classifiées du Tableau 8.1 à la page 163. Ensuite, l'agent constitue un mappage aussi bon que possible du vecteur de caractéristique à la valeur de fonction (par exemple la classe de marchandises).

On peut maintenant tenter d'aborder une définition du terme « machine learning ».
Tom Mitchell [Mit97] donne cette définition : Machine
Learning est l'étude d'algorithmes informatiques qui améliorent l'automatisation
principalement par l'expérience.
Sur cette base, nous donnons

> Définition 8.1 Un agent est un agent apprenant s'il améliore ses performances (mesurées par un critère approprié) sur de nouvelles données inconnues au fil du temps (après avoir vu de nombreux exemples d'apprentissage).

Il est important de tester la capacité de généralisation de l'algorithme d'apprentissage sur des données inconnues, les données de test. Sinon, chaque système qui vient d'enregistrer les données d'entraînement semble fonctionner de manière optimale simplement en appelant les données enregistrées. Un agent d'apprentissage est caractérisé par les termes suivants :
Tâche : la tâche de l'algorithme d'apprentissage est d'apprendre une cartographie. Il peut s'agir par exemple de la correspondance entre la taille et la couleur d'une pomme et sa catégorie de marchandises, mais aussi la correspondance entre les 15 symptômes d'un patient et la décision de retirer ou non son appendice.

Figure 8.5 Exploration de données

Agent variable (plus précisément une classe d'agents) : il s'agit ici de décider avec quel algorithme d'apprentissage on va travailler. Si cela a été choisi, la classe de toutes les fonctions apprenables est déterminée.

Données d'apprentissage (expérience) : les données d'apprentissage (échantillon) contiennent les connaissances que l'algorithme d'apprentissage est censé extraire et apprendre. Lors du choix des données d'entraînement, il faut s'assurer qu'il s'agit d'un échantillon représentatif de la tâche à apprendre.

Données de test : importantes pour évaluer si l'agent formé peut bien généraliser les données de formation aux nouvelles données.

Mesure de performance : pour le dispositif de tri des pommes, le nombre de pommes correctement classées. Nous en avons besoin pour tester la qualité d'un agent. Connaître la mesure de la performance est généralement beaucoup plus facile que de connaître la fonction de l'agent. Par exemple, il est facile de mesurer la performance (temps) d'un coureur de 10 000 mètres. Cependant, cela ne signifie nullement que l'arbitre qui mesure le temps puisse courir aussi vite. L'arbitre ne sait mesurer que la performance, mais pas la « fonction » de l'agent dont il mesure la performance.

Qu'est-ce que l'exploration de données ? La tâche d'une machine d'apprentissage pour extraire des connaissances à partir de données de formation. Souvent, le développeur ou l'utilisateur souhaite que la machine d'apprentissage rende les connaissances extraites lisibles également pour les humains. C'est encore mieux si le développeur peut même modifier les connaissances. Le processus d'induction des arbres de décision dans la Sect. 8.4 est un exemple de ce type de méthode.

Des défis similaires viennent du commerce électronique et de la gestion des connaissances. Une problématique classique se présente ici : à partir des actions des visiteurs sur son portail web,

le propriétaire d'une entreprise Internet souhaite créer une relation entre le
caractéristiques d'un client et la classe de produits qui l'intéressent
client. Ensuite, un vendeur pourra placer des publicités spécifiques au client. Ce
est démontré de manière éloquente sur www.amazon.com, où le client se voit recommander des produits
similaires à ceux vus lors de la visite précédente. Dans de nombreux
Dans les domaines de la publicité et du marketing, ainsi que dans la gestion de la relation client (CRM), les
techniques d'exploration de données commencent à être utilisées. Chaque fois que de grandes quantités
de données sont disponibles, on peut tenter d'utiliser ces données pour l'analyse de la clientèle
préférences afin d'afficher des publicités spécifiques au client. Le domaine émergent
d'apprentissage des préférences est dédié à cet effet.

Le processus d'acquisition de connaissances à partir de données, ainsi que sa représentation et l'application, s'appelle l'exploration de données. Les méthodes utilisées sont généralement prises à partir de statistiques ou d'apprentissage automatique et devrait être applicable à de très grands quantités de données à un coût raisonnable.

Dans le cadre de l'acquisition d'informations, par exemple sur Internet ou dans un intranet, le text mining joue un rôle de plus en plus important. Les tâches typiques incluent la recherche de texte similaire dans un moteur de recherche ou la classification de textes, qui par exemple
est appliqué dans les filtres anti-spam pour les e-mails. Insecte. 8.6.1 nous allons introduire la généralisation algorithme naïf de Bayes pour la classification de texte. Un défi relativement nouveau pour
L'exploration de données est l'extraction d'informations structurelles, statiques et dynamiques à partir de structures graphiques telles que les réseaux sociaux, les réseaux de trafic ou le trafic Internet.

Parce que les deux tâches décrites d'apprentissage automatique et d'exploration de données sont formellement très similaires, les méthodes de base utilisées dans les deux domaines sont pour la plupart identiques. Ainsi dans la description des algorithmes d'apprentissage, aucune distinction ne sera entre l'apprentissage automatique et l'exploration de données.

En raison de l'énorme impact commercial des techniques d'exploration de données, il existe désormais de nombreuses optimisations sophistiquées et toute une gamme de puissants systèmes d'exploration de données, qui offrent une large palette d'outils pratiques pour l'extraction de connaissances à partir de données. Un tel système est introduit dans la Sect. 8.8.

8.1 Analyse des données

Les statistiques offrent plusieurs façons de décrire les données avec des paramètres simples. Depuis nous en choisissons quelques-unes qui sont particulièrement importantes pour l'analyse de la formation données et testez-les sur un sous-ensemble des données LEXMED de la Sect. 7.3. Dans cet exemple ensemble de données, les symptômes x1,...,x15 de N = 473 patients, décrits de manière concise dans le tableau 8.2 à la page 167, ainsi que l'étiquette de classe, c'est-à-dire le diagnostic (appendicite positif/négatif)—sont répertoriés. Le patient numéro un, par exemple, est décrit par le vecteur

$$^{1}x = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37{,}9, 38{,}8, 23100, 0, 1)$$

Tableau 8.2 Description de variables x1,...,x16.

Un peu différent formalisation a été utilisée dans Tableau 7.2 à la page 133

Var. num.	Description	Valeurs
1	Âge	Continu
2	Sexe (1=masculin, 2=féminin)	1, 2
3	Quadrant 1 de la douleur	0, 1
4	Quadrant 2 de la douleur	0, 1
5	Quadrant 3 de la douleur	0, 1
6	Quadrant 4 de la douleur	0, 1
7	Garde musculaire locale	0, 1
8	Défense musculaire généralisée	0, 1
9	Rebond tendresse	0, 1
dix	Douleur au tapotement	0, 1
11	Douleur au toucher rectal	0, 1
12	Température axiale	Continu
13	La température rectale	Continu
14	Leucocytes	Continu
15	Diabète sucré	0, 1
16	Appendicite	0, 1

et le patient numéro deux par

$$^{2}x_{-} = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36{,}9, 37{,}4, 8100, 0, 0)$$

Le patient numéro deux a la valeur leucocytaire $x_{14}^{2} = 8100$.

Pour chaque variable xi , sa moyenne $\bar{x}_i$ est définie comme

$$\bar{x}_i := \frac{1}{N} \sum_{p=1}^{N} x_{i}^{p}$$

et l'écart type si comme mesure de son écart moyen par rapport à la moyenne valeur comme

$$si := \sqrt{\frac{1}{N-1} \sum_{p=1}^{N} (x_{i}^{p} - \bar{x}_i)^2}.$$

La question de savoir si deux variables xi et xj sont statistiquement dépendantes (corrélées) est importante pour l'analyse de données multidimensionnelles. Par exemple, le covariance

$$\sigma_{ij} = \frac{1}{N-1} \sum_{p=1}^{N} (x_{i}^{p} - \bar{x}_i)(x_{j}^{p} - \bar{x}_j)$$

Tableau 8.3 Matrice de corrélation des 16 variables d'appendicite mesurées dans 473 cas

1.	−0,009	0,14	0,037	−0,096	0,12	0,018	0,051	−0,034	−0,041	0,034	0,037	0,05	−0,037	0,37	0,012
−0,009	1.	−0,0074	−0,019	−0,06	0,063	−0,17	0,0084	−0,17	−0,14	−0,13	−0,017	−0,034	−0,14	0,045	−0,2
0,14	−0,0074	1.	0,55	−0,091	0,24	0,13	0,24	0,045	0,18	0,028	0,02	0,045	0,03	0,11	0,045
0,037	−0,019	0,55	1.	−0,24	0,33	0,051	0,25	0,074	0,19	0,087	0,11	0,12	0,11	0,14	−0,0091
−0,096	−0,06	−0,091	−0,24	1.	0,059	0,14	0,034	0,14	0,049	0,057	0,064	0,058	0,11	0,017	0,14
0,12	0,063	0,24	0,33	0,059	1.	0,071	0,19	0,086	0,15	0,048	0,11	0,12	0,063	0,21	0,053
0,018	−0,17	0,13	0,051	0,14	0,071	1.	0,16	0,4	0,28	0,2	0,24	0,36	0,29	−0,0001	0,33
0,051	0,0084	0,24	0,25	0,034	0,19	0,16	1.	0,17	0,23	0,24	0,19	0,24	0,27	0,083	0,084
−0,034	−0,17	0,045	0,074	0,14	0,086	0,4	0,17	1.	0,53	0,25	0,19	0,27	0,27	0,026	0,38
−0,041	−0,14	0,18	0,19	0,049	0,15	0,28	0,23	0,53	1.	0,24	0,15	0,19	0,23	0,02	0,32
0,034	−0,13	0,028	0,087	0,057	0,048	0,2	0,24	0,25	0,24	1.	0,17	0,17	0,22	0,098	0,17
0,037	−0,017	0,02	0,11	0,064	0,11	0,24	0,19	0,19	0,15	0,17	1.	0,72	0,26	0,035	0,15
0,05	−0,034	0,045	0,12	0,058	0,12	0,36	0,24	0,27	0,19	0,17	0,72	1.	0,38	0,044	0,21
−0,037	−0,14	0,03	0,11	0,11	0,063	0,29	0,27	0,27	0,23	0,22	0,26	0,38	1.	0,051	0,44
0,37	0,045	0,11	0,14	0,017	0,21	−0,0001	0,083	0,026	0,02	0,098	0,035	0,044	0,051	1.	−0,0055
0,012	−0,2	0,045	−0,0091	0,14	0,053	0,33	0,084	0,38	0,32	0,17	0,15	0,21	0,44	−0,0055	1.

donne des informations à ce sujet. Dans cette somme, la somme renvoie une entrée positive pour le pième vecteur de données exactement lorsque les écarts des ième et jième composantes par rapport à la moyenne des deux ont le même signe. S'ils ont des signes différents, alors l'entrée est négatif. Par conséquent, la covariance σ 12,13 des deux valeurs de fièvre différentes doit être clairement positif.

Cependant, la covariance dépend aussi de la valeur absolue des variables, ce qui rend difficile la comparaison des valeurs. Pour pouvoir comparer le diplôme de dépendance dans le cas de variables multiples, on définit donc la corrélation coefficient

$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}$$

pour deux valeurs xi et xj , qui n'est rien d'autre qu'une covariance normalisée. La matrice K de tous les coefficients de corrélation contient des valeurs entre −1 et 1, est symétrique, et tous ses éléments diagonaux ont la valeur 1. La matrice de corrélation pour les 16 variables est donnée dans le tableau 8.3.

Cette matrice devient un peu plus lisible lorsqu'on la représente comme une densité parcelle. Au lieu des valeurs numériques, les éléments de la matrice de la Fig. 8.6 à la page 169 sont remplis de valeurs de gris. Dans le diagramme de droite, les valeurs absolues sont indiquées. Ainsi on voit très rapidement quelles variables affichent une dépendance faible ou forte. Nous peut voir, par exemple, que les variables 7, 9, 10 et 14 présentent la plus forte corrélation avec la variable de classe appendicite et sont donc plus importantes pour le diagnostic que l'autre variable. On voit aussi, cependant, que les variables 9 et 10 sont fortement corrélés. Cela pourrait signifier que l'une de ces deux valeurs est potentiellement suffisante pour le diagnostic.

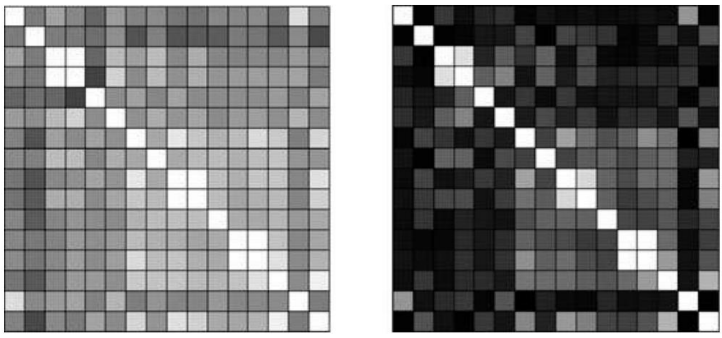
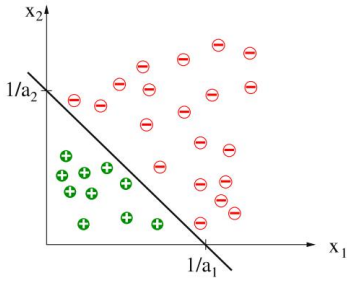


Fig. 8.6 La matrice de corrélation sous forme de graphique de fréquence. Dans le diagramme de gauche, sombre signifie négatif et clair pour positif. Dans l' image de droite, les valeurs absolues étaient répertoriées. Ici noir signifie Kij ≈ 0 (non corrélé) et blanc |Kij | ≈ 1 (fortement corrélé)

Fig. 8.7 Un ensemble de données bidimensionnelles linéairement séparables. L'équation de la droite de division est a1x1 + a2x2 = 1



8.2 Le Perceptron, un classificateur linéaire

Dans l'exemple de classification du tri des pommes, une ligne de séparation courbe est tracée entre les deux classes dans la Fig. 8.3 à la page 163. Un cas plus simple est illustré dans la Fig. 8.7. Ici, les exemples de formation en deux dimensions peuvent être séparés par une ligne droite. Nous appelons un tel ensemble de données d'apprentissage linéairement séparables. En n dimensions, un hyperplan est nécessaire pour la séparation. Cela représente un sous-espace linéaire de dimension n - 1.

Parce que chaque hyperplan (n - 1) dimensionnel dans l'équation R^n peut être décrit par un

$$\sum_{je=1}^{n} aixi = \theta$$

il est logique de définir la séparabilité linéaire comme suit.

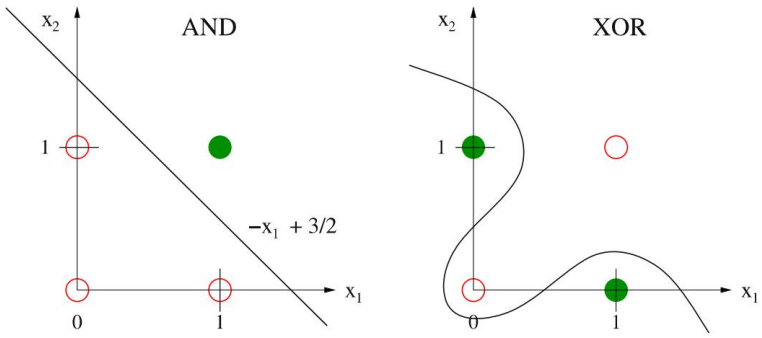


Fig. 8.8 La fonction booléenne AND est linéairement séparable, mais XOR n'est pas (ˆ= true, ○ ˆ= faux)

Définition 8.2 Deux ensembles M1 R n et M2 R n sont dits linéairement séparables si des nombres réels a1,...,an, θ existent avec

$$\sum_{i=1}^{n} a_i x_i > \theta \text{ pour tout } x \in M1 \text{ et } \sum_{i=1}^{n} a_i x_i \leq \theta \text{ pour tout } x \in M2.$$

La valeur θ est notée seuil.

Dans la Fig. 8.8 , nous voyons que la fonction AND est linéairement séparable, mais la fonction XOR ne l'est pas. Pour AND, par exemple, la ligne −x1 + 3/2 sépare les interprétations vraies et fausses de la formule x1 x2. En revanche, la fonction XOR n'a pas de ligne droite de séparation. Il est clair que la fonction XOR a une structure plus complexe que la fonction AND à cet égard.

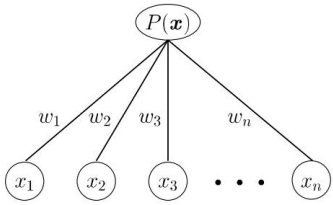
Avec le perceptron, nous présentons un algorithme d'apprentissage très simple qui peut séparer arate des ensembles linéairement séparables.

Définition 8.3 Soit w = (w1,...,wn) R un vecteur n un vecteur de poids et x R →{0, n d'entrée. Un perceptron représente une fonction P : R correspond à la n 1} qui règle suivante :

$$P(x) = \begin{cases} 1 & \text{si } wx = \sum_{i=1}^{n} w_i x_i > 0, \\ 0 & \text{autre.} \end{cases}$$

Le perceptron [Ros58, MP69] est un algorithme de classification très simple. Il équivaut à un réseau de neurones à deux couches avec activation par une fonction de seuil, illustré à la Fig. 8.9 à la page 171. Comme indiqué au Chap. 9, chaque nœud du réseau représente un neurone et chaque bord une synapse. Pour l'instant, cependant, nous allons seulement

Fig. 8.9 Représentation graphique d'un perceptron sous la forme d'un réseau de neurones à deux couches



considérez le perceptron comme un agent d'apprentissage, c'est-à-dire comme une fonction mathématique qui associe un vecteur de caractéristiques à une valeur de fonction. Ici, les variables d'entrée xi sont notées caractéristiques.

Comme nous pouvons le voir dans la formule $\sum_{i=1}^{n} w_i x_i > 0$, tous les points x au-dessus de l'hyperplan $\sum_{i=1}^{n} w_i x_i = 0$ sont classés comme positifs (P (x) = 1), et tous les autres comme négatifs (P (x) = 0). L'hyperplan séparateur passe par l'origine car θ = 0.
Nous utiliserons une petite astuce pour montrer que l'absence de seuil arbitraire ne représente aucune restriction de puissance. Cependant, nous voulons d'abord introduire un algorithme d'apprentissage simple pour le perceptron.

8.2.1 La règle d'apprentissage

Avec la notation M+ et M− pour les ensembles de modèles d'apprentissage positifs et négatifs respectivement, la règle d'apprentissage du perceptron se lit [MP69]

```
PERCEPTRONLEARNING[M+,M−] w =
vecteur arbitraire de nombres réels
Répéter
    Pour tout x    M+
        Si wx ≤ 0 Alors w = w + x
    Pour tout x    M−
        Si wx > 0 Alors w = w − x
Jusqu'à ce que tous les x    M+    M− soient correctement classés
```

Le perceptron devrait sortir la valeur 1 pour tout x M+. D'après la définition 8.3 à la page 170 , cela est vrai lorsque wx > 0. Si ce n'est pas le cas, alors x est ajouté au vecteur de poids w, le vecteur de poids étant modifié exactement dans la bonne direction.
Nous le voyons lorsque nous appliquons le perceptron au vecteur modifié w + x car

$$2\,(w + x) \cdot x = wx + x \qquad \cdot$$

Si cette étape est répétée assez souvent, à un moment donné, la valeur wx deviendra positive, comme il se doit. De manière analogue, on voit que, pour des données d'entraînement négatives, le

perceptron calcule une valeur de plus en plus petite

$$(w - x) \cdot x = wx - x^2$$

qui à un moment donné devient négatif.[2]

Exemple 8.2 Un perceptron est à entraîner sur les ensembles M+ = {(0, 1.8), (2, 0.6)} et M− = {(−1.2, 1.4), (0.4,−1)}. w = (1, 1) a été utilisé comme vecteur de poids initial. Les données d'apprentissage et la ligne définie par le vecteur de poids wx = x1 + x2 = 0 sont illustrées à la Fig. 8.10 à la page 173 dans la première image de la rangée supérieure. De plus, le vecteur de poids est dessiné sous la forme d'une ligne en pointillés. Comme wx = 0, c'est orthogonal à la droite.

Dans la première itération à travers la boucle de l'algorithme d'apprentissage, la seule fausse exemple de formation classifié est (−1.2, 1.4) parce que

$$(-1.2, 1.4) \cdot \begin{matrix} 1 \\ 1 \end{matrix} = 0{,}2 > 0.$$

Il en résulte w = (1, 1)−(−1.2, 1.4) = (2.2,−0.4), comme illustré dans la deuxième image de la rangée supérieure de la Fig. 8.10 à la page 173. Les autres images montrent comment, après un total de cinq changements, la ligne de démarcation se situe entre les deux classes. Le perceptron classe ainsi correctement toutes les données. Nous voyons clairement dans l'exemple que chaque point de données mal classé de M+ « tire » le vecteur de poids w dans sa direction et chaque point mal classé de M− « pousse » le vecteur de poids dans la direction opposée.

Il a été montré [MP69] que le perceptron converge toujours pour sep linéairement données arables. Nous avons

Théorème 8.1 Soient les classes M+ et M− linéairement séparables par un hyperplan wx = 0. Alors PERCEPTRONLEARNING converge pour chaque initialisation du vecteur w. Le perceptron P avec le vecteur poids ainsi calculé divise les classes M+ et M−, soit :

$$P(x) = 1 \quad x \quad M+$$

et

$$P(x) = 0 \quad x \quad M-.$$

Comme nous pouvons le voir clairement dans l'exemple 8.2, les perceptrons tels que définis ci-dessus ne peuvent diviser arbitrairement des ensembles linéairement séparables, mais seulement ceux qui sont divisibles par une droite passant par l'origine, ou dans R^n par un hyperplan passant par l'origine, car le terme constant θ est absent de l'équation $\sum_{i=1}^{n} w_i x_i = 0$.

2Attention ! Ceci n'est pas une preuve de convergence pour la règle d'apprentissage du perceptron. Cela montre seulement que le perceptron converge lorsque l'ensemble de données d'apprentissage se compose d'un seul exemple.

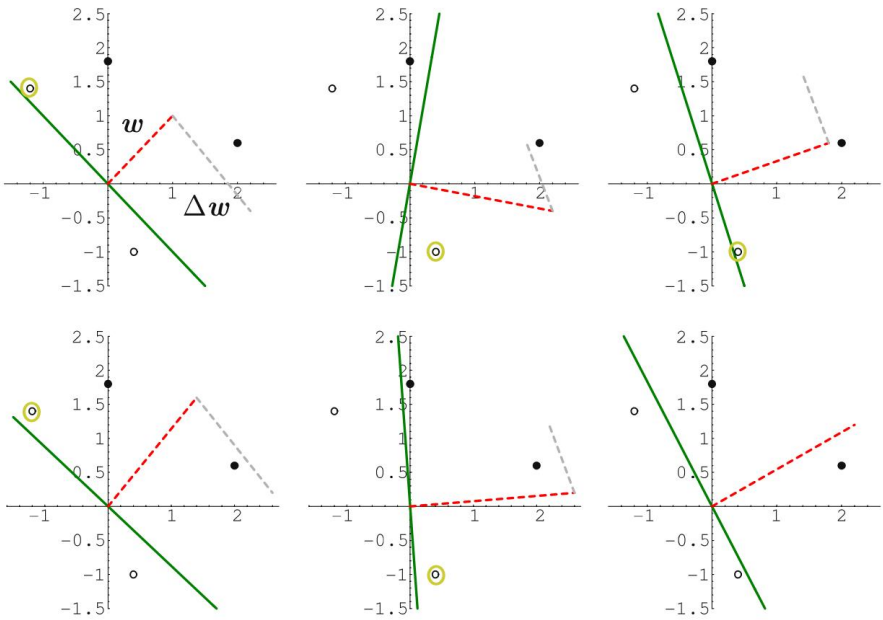


Fig. 8.10 Application de la règle d'apprentissage du perceptron à deux points de données positifs (•) et deux négatifs (◦). La ligne continue montre la ligne de division actuelle wx = 0. La ligne pointillée orthogonale est le vecteur de poids w et la deuxième ligne pointillée le vecteur de changement w = x ou w = −x à ajouter, qui est calculé à partir du point de données actuellement actif entouré de gris

Avec l'astuce suivante, nous pouvons générer le terme constant. Nous maintenons la dernière composante xn du vecteur d'entrée x constante et lui donnons la valeur 1. Maintenant, le poids wn = : −θ fonctionne comme un seuil car

$$\sum_{je=1}^{n} wxi = \sum_{je=1}^{n-1} wixi - \theta > 0 \qquad \sum_{je=1}^{n-1} wixi > \theta \ .$$

Une telle valeur constante xn = 1 dans l'entrée est appelée unité de biais. Étant donné que le poids associé provoque un déplacement constant de l'hyperplan, le terme «biais» convient bien.

Dans l'application de l'algorithme d'apprentissage du perceptron, un bit avec la valeur constante 1 est ajouté au vecteur de données d'apprentissage. On observe que le poids wn, ou le seuil θ , est appris au cours du processus d'apprentissage.

Or il a été montré qu'un perceptron $P\theta : R^{n-1} \rightarrow \{0, 1\}$

$$P\theta\ (x1,...,xn-1) = \begin{cases} 1 \text{ si } \sum_{i=1}^{n-1} wixi > \theta \ , \\ 0 \text{ autre} \end{cases} \qquad (8.1)$$

avec un seuil arbitraire peut être simulé par un perceptron $P : R^{n} \rightarrow \{0, 1\}$ le seuil avec 0. Si on compare (8.1) avec la définition de linéairement séparable, alors on voit que les deux énoncés sont équivalents. En résumé, nous avons montré que :

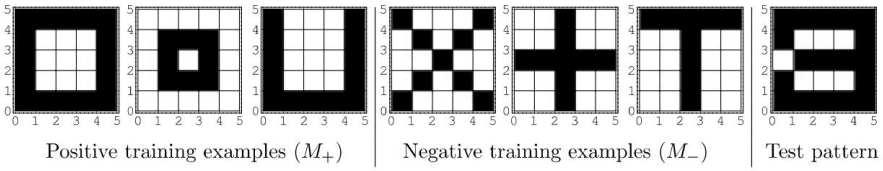
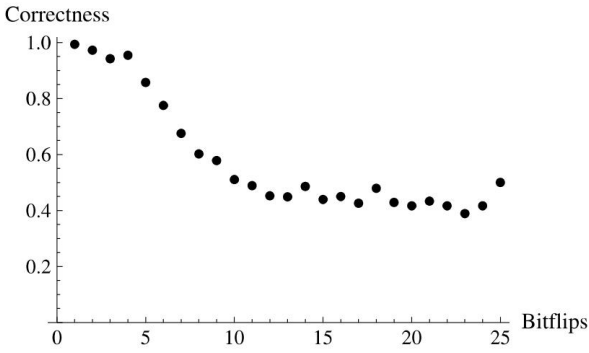


Fig. 8.11 Les six modèles utilisés pour l'entraînement. Le motif tout droit est l'un des 22 motifs de test pour le premier motif avec une séquence de quatre bits inversés

Fig. 8.12 Exactitude relative du perceptron en fonction du nombre de bits inversés dans les données de test



Théorème 8.2 Une fonction f : R →{0, 1} peut être représentée par un perceptron si et seulement si les deux ensembles de vecteurs d'entrée positifs et négatifs sont linéairement séparables.

Exemple 8.3 Nous entraînons maintenant un perceptron avec un seuil sur six motifs binaires graphiques simples, représentés sur la Fig. 8.11, avec 5 × 5 pixels chacun.

Les données d'apprentissage peuvent être apprises par PERCEPTRONLEARNING en quatre itérations sur tous les modèles. Des modèles avec un nombre variable de bits inversés introduits comme bruit sont utilisés pour tester la capacité de généralisation du système. Les bits inversés dans le motif de test sont dans chaque cas en séquence l'un après l'autre. Sur la Fig. 8.12 , le pourcentage de motifs correctement classés est tracé en fonction du nombre de faux bits.

Après environ cinq bits inversés consécutifs, l'exactitude chute brusquement, ce qui n'est pas surprenant compte tenu de la simplicité du modèle. Dans la section suivante, nous présenterons un algorithme qui fonctionne beaucoup mieux dans ce cas.

8.2.2 Optimisation et perspectives

En tant que l'un des algorithmes d'apprentissage basés sur les réseaux de neurones les plus simples, le perceptron à deux couches ne peut diviser que des classes séparables linéairement. Insecte. 9.5, nous verrons que les réseaux multicouches sont nettement plus puissants. Malgré sa structure simple, le

perceptron sous la forme présentée converge très lentement. Elle peut être accélérée par la normalisation du vecteur de modification du poids. Les formules w = w ± x sont remplacées par w = w ± x/|x|. Ainsi, chaque point de données a le même poids lors de l'apprentissage, indépendamment de sa valeur.

La vitesse de convergence dépend fortement de l'initialisation du vecteur w. Idéalement, il n'aurait pas besoin d'être modifié du tout et l'algorithme convergerait après une itération. On peut se rapprocher de cet objectif en utilisant l'initialisation heuristique

$$w0 = \frac{x-}{x\ M+} - \frac{X,}{x\ M-}$$

que nous étudierons plus en détail dans l'exercice 8.5 à la page 217.

Si nous comparons la formule du perceptron avec la méthode de notation présentée dans la Sect. 7.3.1, on voit immédiatement leur équivalence. De plus, le perceptron, en tant que modèle de réseau de neurones le plus simple, est équivalent au Bayes naïf, le type le plus simple de réseau bayésien (voir l'exercice 8.16 à la page 219). Ainsi évidemment plusieurs algorithmes de classification très différents ont une origine commune.

Au Chap. 9 nous allons nous familiariser avec une généralisation du perceptron sous la forme de l'algorithme de rétropropagation, qui peut diviser des ensembles non linéairement séparables grâce à l'utilisation de plusieurs couches, et qui possède une meilleure règle d'apprentissage.

8.3 La méthode du plus proche voisin

Pour un perceptron, les connaissances disponibles dans les données d'apprentissage sont extraites et enregistrées sous une forme compressée dans les poids wi . Ainsi, les informations sur les données sont perdues. C'est exactement ce qui est souhaité, cependant, si le système est censé généraliser des données d'apprentissage à de nouvelles données. Dans ce cas, la généralisation est un processus chronophage dont le but est de trouver une représentation compacte des données sous la forme d'une fonction qui classe au mieux les nouvelles données.

La mémorisation de toutes les données en les sauvegardant simplement est tout autre. Ici, l'apprentissage est extrêmement simple. Cependant, comme mentionné précédemment, les connaissances sauvegardées ne sont pas si facilement applicables à de nouveaux exemples inconnus. Une telle approche est très inadaptée à l'apprentissage du ski par exemple. Un débutant ne peut jamais devenir un bon skieur simplement en regardant des vidéos de bons skieurs. Évidemment, lorsque des mouvements d'apprentissage de ce type sont exécutés automatiquement, quelque chose de similaire se produit comme dans le cas du perceptron. Après une pratique suffisamment longue, les connaissances stockées dans les exemples d'entraînement sont transformées en une représentation interne dans le cerveau.

Cependant, il existe des exemples réussis de mémorisation dans lesquels la généralisation est également possible. Lors du diagnostic d'un cas difficile, un médecin pourrait essayer de se souvenir de cas similaires du passé. Si sa mémoire est bonne, alors il pourrait tomber sur ce cas, le rechercher dans ses dossiers et finalement arriver à un diagnostic similaire. Pour cette approche, le médecin doit avoir un bon sens de la similitude, afin de se souvenir du cas le plus similaire. S'il a trouvé cela, alors il doit se demander si c'est assez similaire pour justifier le même diagnostic.

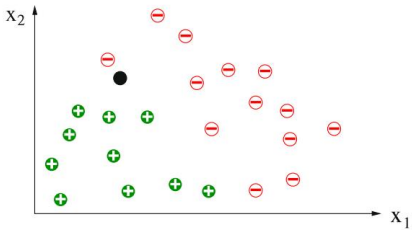


Fig. 8.13 Dans cet exemple avec des exemples d'apprentissage négatifs et positifs, la méthode du plus proche voisin regroupe le nouveau point marqué en noir dans la classe négative

Que signifie similarité dans le contexte formel que nous construisons ? Nous représentons les échantillons d'apprentissage comme d'habitude dans un espace de caractéristiques multidimensionnel et définissons : Plus leur distance dans l'espace de caractéristiques est petite, plus deux exemples sont similaires.

Nous appliquons maintenant cette définition à l'exemple bidimensionnel simple de la Fig. 8.13. Ici, le prochain voisin du point noir est un exemple négatif. Il est donc affecté à la classe négative.

La distance d(x, y) entre deux points x R mesurée n et y R n peut par exemple être par la distance euclidienne

$$d(x, y) = |x - y| = \sqrt{\sum_{je=1}^{n} (xi - yi)^2}.$$

Comme il existe de nombreuses autres mesures de distance en plus de celle-ci, il est logique de réfléchir à des alternatives pour une application concrète. Dans de nombreuses applications, certaines fonctionnalités sont plus importantes que d'autres. Par conséquent, il est souvent judicieux de mettre à l'échelle les caractéristiques différemment selon les poids wi : La formule se lit alors

$$dw(x, y) = |x - y| = \sqrt{\sum_{je=1}^{n} wi(xi - yi)^2}.$$

Le programme simple de classification du plus proche voisin suivant recherche dans les données d'apprentissage le voisin le plus proche t du nouvel exemple s, puis classe s exactement 3 comme t.

```
VOISINPROCHE[M+,M−,s] t =
    argminx   M+   M− {d(s, x)}
    Si t     M+ Alors Retour („+")
                  Sinon Retour(„–")
```

3Les fonctionnelles argmin et argmax déterminent, comme min et max, le minimum ou le maximum d'un ensemble ou d'une fonction. Cependant, plutôt que de renvoyer la valeur du maximum ou du minimum, ils donnent la position, c'est-à-dire l'argument dans lequel l'extremum apparaît.

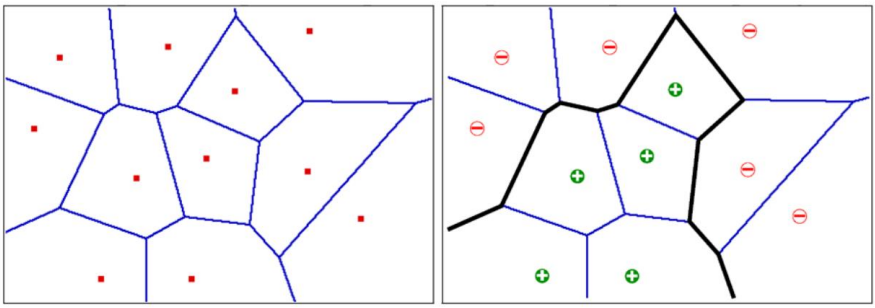


Fig. 8.14 Un ensemble de points avec leur diagramme de Voronoi (à gauche) et la ligne de division générée pour les deux classes M+ et M−

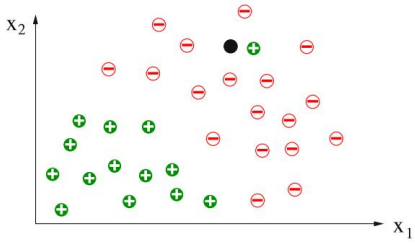


Fig. 8.15 La méthode du voisin le plus proche affecte le nouveau point marqué en noir à la mauvaise classe (positive) car le voisin le plus proche est très probablement mal classé

Contrairement au perceptron, la méthode du plus proche voisin ne génère pas de ligne qui divise les points de données d'apprentissage. Cependant, une ligne imaginaire séparant les deux classes existe certainement. Nous pouvons générer cela en générant d'abord le soi-disant diagramme de Voronoi. Dans le diagramme de Voronoi, chaque point de données est entouré d'un polygone convexe, qui définit ainsi un voisinage autour de lui. Le diagramme de Voronoi a la propriété que pour un nouveau point arbitraire, le voisin le plus proche parmi les points de données est le point de données, qui se trouve dans le même voisinage. Si le diagramme de Voronoi pour un ensemble de données d'apprentissage est déterminé, il est alors simple de trouver le voisin le plus proche pour un nouveau point à classer. L'appartenance à la classe sera alors prélevée sur le voisin le plus proche.

Sur la figure 8.14 , nous voyons clairement que la méthode du plus proche voisin est significativement plus puissante que le perceptron. Il est capable de réaliser correctement des lignes de partage arbitrairement complexes (en général : des hyperplans). Cependant, il y a un danger ici. Un seul point erroné peut dans certaines circonstances conduire à de très mauvais résultats de classement. Un tel cas se produit sur la Fig. 8.15 lors de la classification du point noir. La méthode du voisin le plus proche peut classer cette erreur. Si le point noir est immédiatement à côté d'un point positif qui est une valeur aberrante de la classe positive, alors il sera classé positif plutôt que négatif comme cela serait prévu ici. Un ajustement erroné à des erreurs aléatoires (bruit) est appelé surajustement.

K-VOISIN LE PLUS PROCHE(M+,M−,s)

V = {k plus proches voisins dans M+ M−}
Si |M+ ∩V | > |M− ∩V | Puis Retour(„+”)
SinonSi |M+ ∩V | < |M− ∩V | Puis Retour(„–”)
Autre Retour(Aléatoire(„+”, „–”))

Fig. 8.16 L' ALGORITHME K-NEARESTNEIGHBOR

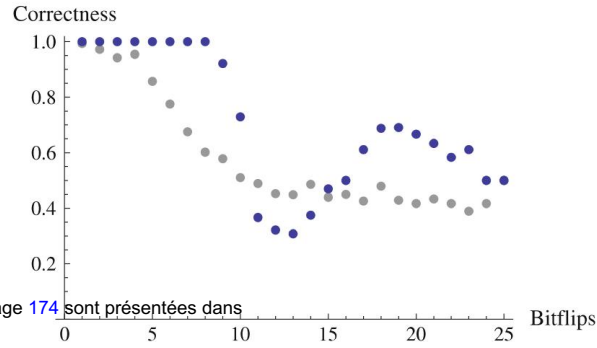


Fig. 8.17 Exactitude relative de la classification du plus proche voisin en fonction du nombre de bits inversés. La structure de la courbe avec son minimum à 13 et son maximum à 19 est liée à la structure particulière des données d'apprentissage. A titre de comparaison, les valeurs du perceptron de l'exemple 8.3 à la page 174 sont présentées dans gris

Pour éviter les fausses classifications dues à des valeurs aberrantes uniques, il est recommandé de lisser quelque peu la surface de division. Cela peut être accompli, par exemple, avec l' algorithme K-NEARESTNEIGHBOR de la Fig. 8.16, qui prend une décision majoritaire parmi les k plus proches voisins.

Exemple 8.4 Nous appliquons maintenant NEARESTNEIGHBOR à l'exemple 8.3 à la page 174. Comme nous avons affaire à des données binaires, nous utilisons la distance de Hamming comme métrique de distance.4 Comme exemple de test, nous utilisons à nouveau des exemples d'apprentissage modifiés avec n bits inversés consécutifs chacun. Sur la Fig. 8.17, le pourcentage d'exemples de test correctement classés est représenté en fonction du nombre de bits inversés b. Jusqu'à huit bits inversés, tous les modèles sont correctement identifiés. Passé ce point, le nombre d'erreurs augmente rapidement. Ceci n'est pas surprenant car le modèle d'entraînement numéro 2 de la Fig. 8.11 à la page 174 de la classe M+ a une distance de Hamming de 9 par rapport aux deux exemples d'entraînement, les numéros 4 et 5 de l'autre classe. Cela signifie que le motif de test est très probablement proche des motifs de l'autre classe. On voit assez clairement que la classification du plus proche voisin est supérieure au perceptron dans cette application jusqu'à huit faux bits.

4La distance de Hamming entre deux vecteurs de bits est le nombre de bits différents des deux vecteurs.

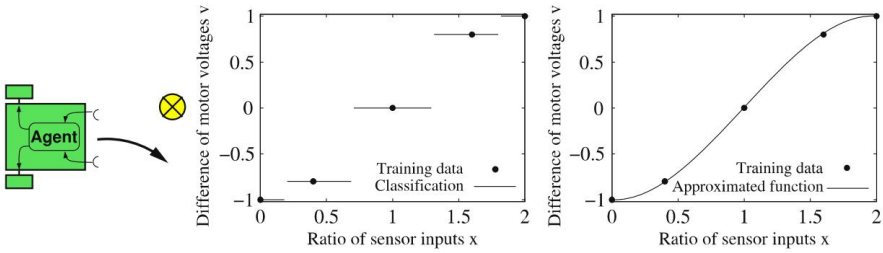


Fig. 8.18 L'agent d'apprentissage, qui est censé éviter la lumière (à gauche), représenté comme un classifieur (au milieu), et comme une approximation (à droite)

8.3.1 Deux classes, plusieurs classes, approximation

La classification du voisin le plus proche peut également être appliquée à plus de deux classes. Tout comme dans le cas de deux classes, la classe du vecteur de caractéristiques à classer est simplement définie comme la classe du voisin le plus proche. Pour la méthode des k plus proches voisins, la classe doit être déterminée comme la classe ayant le plus de membres parmi les k plus proches voisins.

Si le nombre de classes est important, il n'est généralement plus judicieux d'utiliser des algorithmes de classification car la taille des données d'apprentissage nécessaires augmente rapidement avec le nombre de classes. En outre, dans certaines circonstances, des informations importantes sont perdues lors du classement de nombreuses classes. Cela deviendra clair dans l'exemple suivant.

Exemple 8.5 Un robot autonome avec des capteurs simples similaires aux véhicules Braitenberg présentés dans la Fig. 1.1 à la page 2 est censé apprendre à s'éloigner de la lumière. Cela signifie qu'il doit apprendre de manière aussi optimale que possible à mapper ses valeurs de capteur sur un signal de direction qui contrôle la direction de conduite. Le robot est équipé de deux capteurs de lumière simples sur sa face avant. A partir des deux signaux de capteur (avec sl pour le capteur gauche et sr pour le capteur droit), la relation x = sr/sl est calculée.
Pour commander les moteurs électriques des deux roues à partir de cette valeur x, la différence v = Ur - Ul des deux tensions Ur et Ul des moteurs gauche et droit, respectivement.
La tâche de l'agent d'apprentissage est maintenant d'éviter un signal lumineux. Il doit donc apprendre une 5 application f qui calcule la valeur « correcte » v = f (x).

Pour cela, nous réalisons une expérience dans laquelle, pour quelques valeurs mesurées x, nous trouvons une valeur v aussi optimale que possible. Ces valeurs sont tracées sous forme de points de données sur la Fig. 8.18 et doivent servir de données d'apprentissage pour l'agent d'apprentissage. Lors de la classification du voisin le plus proche, chaque point de l'espace des caractéristiques (c'est-à-dire sur l'axe des x) est classé exactement comme son voisin le plus proche parmi les données d'apprentissage. La fonction de pilotage des moteurs est alors une fonction échelon à grands sauts (Fig. 8.18 milieu). Si nous voulons des étapes plus fines, nous devons fournir en conséquence plus de données d'entraînement

5Pour que l'exemple reste simple et lisible, le vecteur caractéristique x a été délibérément conservé dimensionnel.

d'autre part, nous pouvons obtenir une fonction continue si nous approchons une fonction lisse pour ajuster les cinq points (Fig. 8.18 à la page 179 à droite). Exiger que la fonction f soit continue conduit à de très bons résultats, même sans points de données supplémentaires.

Pour l'approximation de fonctions sur des points de données, il existe de nombreuses méthodes mathématiques, telles que l'interpolation polynomiale, l'interpolation spline ou la méthode des moindres carrés. L'application de ces méthodes devient problématique en dimension supérieure. La difficulté particulière de l'IA est que des méthodes d'approximation sans modèle sont nécessaires. C'est-à-dire qu'une bonne approximation des données doit être faite sans connaissance des propriétés particulières des données ou de l'application. De très bons résultats ont été obtenus ici avec les réseaux de neurones et autres approximateurs de fonctions non linéaires, qui sont présentés au Chap. 9.

La méthode des k plus proches voisins peut être appliquée de manière simple au problème d'approximation. Dans l'algorithme K-NEARESTNEIGHBOR, après que l'ensemble V = {x1, x2,..., xk } est déterminé, la valeur de la fonction moyenne des k plus proches voisins

$$f(\hat{x}) = \frac{1}{k} \sum_{je=1}^{k} f(xi) \tag{8.2}$$

est calculé et pris comme approximation f^ pour le vecteur requête x. Plus k devient grand, plus la fonction f^ est lisse.

8.3.2 La distance est pertinente

Dans l'application pratique des variantes discrètes et continues de la méthode des k plus proches voisins, des problèmes surviennent souvent. Lorsque k devient grand, il existe généralement plus de voisins à grande distance que de voisins à petite distance. Ainsi le calcul de f^ est dominé par des voisins éloignés. Pour éviter cela, les k voisins sont pondérés de manière à ce que les voisins les plus éloignés aient moins d'influence sur le résultat. Lors de la décision majoritaire dans l'algorithme K-NEARESTNEIGHBOR, les "votes" sont pondérés avec le poids

$$wi = \frac{1}{1 + ad(x, xi)^2}, \tag{8.3}$$

qui diminue avec le carré de la distance. La constante α détermine la vitesse de décroissance des poids. L'équation (8.2) est maintenant remplacée par

$$f(\hat{x}) = \frac{\sum_{Ki=1} wif(xi)}{\sum_{ki=1} wi}.$$

Pour une concentration uniformément distribuée de points dans l'espace des caractéristiques, cela garantit que l'influence des points s'approche asymptotiquement de zéro à mesure que la distance augmente. Ainsi, il devient possible d'utiliser de nombreuses ou même toutes les données d'apprentissage pour classer ou approximer un vecteur d'entrée donné.

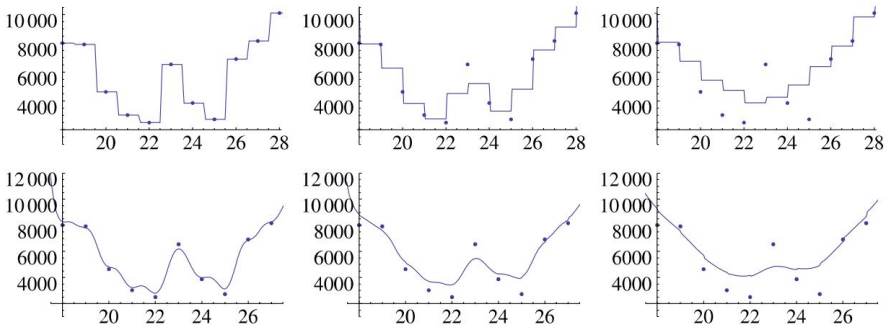


Fig. 8.19 Comparaison de la méthode des k plus proches voisins (ligne supérieure) avec k = 1 (gauche), k = 2 (milieu) et k = 6 (droite), à sa variante pondérée par la distance (ligne inférieure) avec α = 20 (à gauche), α = 4 (au milieu) et α = 1 (à droite) sur un jeu de données unidimensionnel

Pour avoir une idée de ces méthodes, sur la Fig. 8.19 , la méthode du k plus proche voisin (dans la rangée supérieure) est comparée à son optimisation pondérée par la distance. En raison du moyennage, les deux méthodes peuvent généraliser, ou en d'autres termes, annuler le bruit, si le nombre de voisins pour le k plus proche voisin ou le paramètre α est défini de manière appropriée. Les diagrammes montrent bien que la méthode pondérée par la distance donne une approximation beaucoup plus lisse que le k plus proche voisin. En ce qui concerne la qualité de l'approximation, cette méthode très simple peut bien rivaliser avec des algorithmes d'approximation sophistiqués tels que les réseaux de neurones non linéaires, les machines à vecteurs de support et les processus gaussiens.

Il existe de nombreuses alternatives à la fonction de pondération (également appelée noyau) donnée dans (8.3) à la page 180. Par exemple, une fonction gaussienne ou similaire en forme de cloche peut être utilisée. Pour la plupart des applications, les résultats ne sont pas très sensibles sur la sélection du noyau. Cependant, le paramètre de largeur α qui pour toutes ces fonctions doit être défini manuellement a une grande influence sur les résultats, comme le montre la Fig. 8.19. Pour éviter une telle adaptation manuelle gênante, des méthodes d'optimisation ont été développées pour régler automatiquement ce paramètre [SA94, SE10].

8.3.3 Temps de calcul

Comme mentionné précédemment, la formation est accomplie dans toutes les variantes de la méthode du plus proche voisin en sauvegardant simplement tous les vecteurs de formation avec leurs étiquettes (valeurs de classe) ou la valeur de fonction f (x). Ainsi, aucun autre algorithme d'apprentissage n'apprend aussi rapidement. Cependant, répondre à une requête de classification ou d'approximation d'un vecteur x peut devenir très coûteux. Le simple fait de trouver les k voisins les plus proches pour n données d'apprentissage nécessite un coût qui croît linéairement avec n. Pour la classification ou l'approximation, il y a en plus un coût qui est linéaire en k. Le temps de calcul total croît donc comme Θ(n + k). Pour de grandes quantités de données de formation, cela peut entraîner des problèmes.

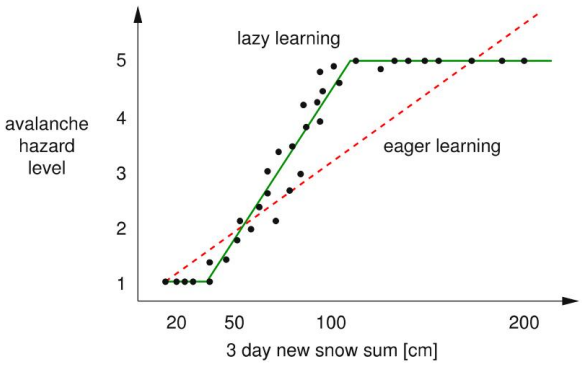


Fig. 8.20 Pour déterminer le risque d'avalanche, une fonction est approximée à partir des données d'entraînement. Voici à titre de comparaison un modèle local (ligne continue) et un modèle global (ligne pointillée)

8.3.4 Résumé et perspectives

Parce que rien ne se passe dans la phase d'apprentissage des méthodes du plus proche voisin présentées, de tels algorithmes sont également appelés apprentissage paresseux, contrairement à l'apprentissage avide, dans lequel la phase d'apprentissage peut être coûteuse, mais l'application à de nouveaux exemples est très efficace. Le perceptron et tous les autres réseaux de neurones, l'apprentissage par arbre de décision et l'apprentissage des réseaux bayésiens sont des méthodes d'apprentissage avides. Étant donné que les méthodes d'apprentissage paresseux ont besoin d'accéder à la mémoire avec toutes les données d'apprentissage pour approximer une nouvelle entrée, elles sont également appelées apprentissage basé sur la mémoire.

Pour comparer ces deux classes de processus d'apprentissage, nous utiliserons comme exemple la tâche consistant à déterminer le risque d'avalanche actuel à partir de la quantité de neige fraîchement tombée dans une certaine région de Suisse.6 Dans la Fig . nous voulons utiliser comme données d'entraînement. Lors de l'application d'un algorithme d'apprentissage avide qui entreprend une approximation linéaire des données, la ligne pointillée représentée sur la figure est calculée. En raison de la limitation à une ligne droite, l'erreur est relativement importante avec un maximum d'environ 1,5 niveaux de risque. Pendant l'apprentissage paresseux, rien n'est calculé avant qu'une requête pour le niveau de danger actuel n'arrive.

Ensuite, la réponse est calculée à partir de plusieurs voisins les plus proches, c'est-à-dire localement. Cela pourrait entraîner la courbe illustrée sur la figure, qui est constituée de segments de ligne et montre des erreurs beaucoup plus petites. L'avantage de la méthode paresseuse est sa localité. L'approximation est prise localement à partir du nouveau niveau de neige actuel et non globalement.
Ainsi pour des classes de fonctions fondamentalement égales (par exemple des fonctions linéaires), les algorithmes paresseux sont meilleurs.

Les méthodes du plus proche voisin sont bien adaptées à toutes les situations problématiques dans lesquelles une bonne approximation locale est nécessaire, mais qui n'imposent pas une exigence élevée sur la vitesse du système. Le prédicteur d'avalanche mentionné ici, qui fonctionne une fois par jour, est une telle application. Les méthodes du plus proche voisin ne conviennent pas lorsqu'une description de la connaissance extraite des données doit être compréhensible par l'homme, ce qui est aujourd'hui le cas pour de nombreuses applications de data mining (voir section 8.4) . Dans

6Le total des chutes de neige sur trois jours est en fait une caractéristique importante pour déterminer le niveau de danger. En pratique, cependant, des attributs supplémentaires sont utilisés [Bra01]. L'exemple utilisé ici est simplifié.

Fonctionnalité	Requête	Cas de la base de cas
Pièce défectueuse :	Feu arrière	Lumière de devant
Modèle de vélo :	Montagne du Pin Marin	VSF T400
Année:	1993	2001
Source d'énergie:	Batterie	Dynamo
État de l'ampoule :	ok	d'accord
État du câble lumineux :	?	d'accord
	Solution	
Diagnostic:	?	Contact électrique avant manquant
Réparation:	?	Établir le contact électrique avant

Fig. 8.21 Exemple de diagnostic simple pour une requête et le cas correspondant de la base de cas

Ces dernières années, ces méthodes d'apprentissage basées sur la mémoire deviennent populaires et diverses variantes améliorées (par exemple la régression linéaire pondérée localement) ont été appliqué [Cle79].

Pour pouvoir utiliser les méthodes décrites, les données de formation doivent être disponibles sous forme de vecteurs d'entiers ou de nombres réels. Ils sont donc inadaptés à applications dans lesquelles les données sont représentées symboliquement, par exemple en premier formules logiques d'ordre. Nous allons maintenant en discuter brièvement.

8.3.5 Raisonnement par cas

Dans le raisonnement par cas (CBR), la méthode du plus proche voisin est étendue aux descriptions de problèmes symboliques et à leurs solutions. Le CBR est utilisé dans le diagnostic de problèmes techniques en service client ou pour les hotlines téléphoniques. L'exemple montré dans la Fig. 8.21 sur le diagnostic d'extinction d'un feu de vélo illustre ce type de situation.

Une solution est recherchée pour la requête d'un client avec un vélo arrière défectueux lumière. Dans la colonne de droite, un cas similaire à la requête dans la colonne du milieu est donné. Cela découle de la base de cas, qui correspond aux données de formation au plus proche méthode voisine. Si nous prenions simplement le cas le plus similaire, comme nous le faisons dans le cas le plus proche méthode voisine, alors nous finirions par essayer de réparer la lumière avant lorsque le le feu arrière est cassé. Nous avons donc besoin d'une transformation inverse de la solution du problème similaire découvert vers la requête. Les étapes les plus importantes de la solution à un cas CBR sont effectuées dans la Fig. 8.22 à la page 184. La transformation dans ce cas L'exemple est simple : le feu arrière est mappé sur le feu avant.

Aussi belle et simple que cette méthode puisse paraître en théorie, en pratique la construction de systèmes de diagnostic CBR est une tâche très difficile. Les trois principales difficultés sont:

Modélisation Les domaines de l'application doivent être modélisés dans un contexte formel.
Ici la monotonie logique, que nous connaissons du Chap. 4, présente des difficultés. Peut le développeur prédit et cartographie tous les cas particuliers possibles et les variantes de problème ?

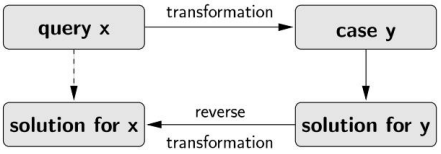


Fig. 8.22 Si pour un cas x un cas similaire y est trouvé, alors pour obtenir une solution pour x, la transformation doit être déterminée et son inverse appliqué au cas découvert y

Similitude Trouver une métrique de similarité appropriée pour les fea symboliques et non numériques tures.

Transformation Même si un cas similaire est trouvé, il n'est pas encore clair à quoi doivent ressembler le mappage de transformation et son inverse.

En effet, il existe aujourd'hui des systèmes CBR pratiques pour les applications de diagnostic. Cependant, pour les raisons évoquées, celles-ci restent loin derrière les experts humains en matière de performance et de flexibilité. Une alternative intéressante au CBR sont les réseaux bayésiens présentés au Chap. 7. Souvent, la représentation symbolique du problème peut également être assez bien mappée à des caractéristiques numériques discrètes ou continues. Ensuite, les méthodes d'apprentissage inductif mentionnées telles que les arbres de décision ou les réseaux de neurones peuvent être utilisées avec succès.

8.4 Apprentissage de l'arbre de décision

L'apprentissage par arbre de décision est un algorithme extrêmement important pour l'IA car il est très puissant, mais aussi simple et efficace pour extraire des connaissances à partir de données. Comparé aux deux algorithmes d'apprentissage déjà introduits, il présente un avantage important. Les connaissances extraites sont non seulement disponibles et utilisables en tant que fonction de boîte noire, mais elles peuvent également être facilement comprises, interprétées et contrôlées par les humains sous la forme d'un arbre de décision lisible. Cela fait également de l'apprentissage par arbre de décision un outil important pour l'exploration de données.

Nous discuterons de la fonction et de l'application de l'apprentissage par arbre de décision à l'aide de l' algorithme C4.5 . C4.5 a été introduit en 1993 par l'Australien Ross Quinlan et est une amélioration de son prédécesseur ID3 (Iterative Dichotomiser 3, 1986). Il est disponible gratuitement pour une utilisation non commerciale [Qui93]. Un autre développement, qui fonctionne encore plus efficacement et peut prendre en compte les coûts des décisions, est C5.0 [Qui93].

Le système CART (Classification and Regression Trees, 1984) développé par Leo Breiman [BFOS84] fonctionne de manière similaire à C4.5. Il a une interface utilisateur graphique pratique, mais il est très coûteux.

Vingt ans plus tôt, en 1964, le système CHAID (Chi-square Automatic Interaction Detectors), capable de générer automatiquement des arbres de décision, a été introduit par J. Sonquist et J. Morgan. Il a la particularité remarquable d'arrêter la croissance de l'arbre avant qu'il ne devienne trop gros, mais aujourd'hui il n'a plus de pertinence.

Tableau 8.4 Variables pour le problème de classification du ski

Variable	Valeur	Description
Ski (objectif variable)	Oui Non	Dois-je conduire jusqu'à la station de ski la plus proche ? avec assez de neige?
Soleil (fonctionnalité)	Oui Non	Y a-t-il du soleil aujourd'hui ?
Snow_Dist (fonctionnalité)	≤100, >100	Distance de la station de ski la plus proche avec bonnes conditions d'enneigement (plus/moins 100 km)
Week-end (fonctionnalité)	Oui Non	C'est le week-end aujourd'hui ?

L'outil d'exploration de données KNIME (Konstanz Information Miner) est également intéressant.
qui a une interface utilisateur conviviale et, en utilisant la bibliothèque Java WEKA , rend également
possibilité d'induction d'arbres de décision. Insecte. 8.8 nous introduirons KNIME.

Maintenant, nous montrons d'abord dans un exemple simple comment un arbre de décision peut être construit
à partir des données d'apprentissage, pour ensuite analyser l'algorithme et l'appliquer au plus
exemple complexe LEXMED pour le diagnostic médical.

8.4.1 Un exemple simple

Un skieur dévoué qui vit près de la haute sierra, une belle chaîne de montagnes en Californie, veut un arbre de
décision pour l'aider à décider s'il vaut la peine de conduire
sa voiture vers une station de ski en montagne. On a donc un problème à deux classes ski oui/non
sur la base des variables répertoriées dans le tableau 8.4.

La Figure 8.23 à la page 186 montre un arbre de décision pour ce problème. Un arbre de décision
est un arbre dont les nœuds internes représentent des caractéristiques (attributs). Chaque arête représente un
valeur d'attribut. À chaque nœud feuille, une valeur de classe est donnée.

Les données utilisées pour la construction de l'arbre de décision sont présentées dans le tableau 8.5 sur
page 186. Chaque ligne du tableau contient les données d'un jour et représente ainsi une
échantillon. En examinant de plus près, nous voyons que la ligne 6 et la ligne 7 se contredisent.
Ainsi, aucun algorithme de classification déterministe ne peut correctement classer toutes les données.
Le nombre de données faussement classées doit donc être ≥1. L'arbre de la Fig. 8.23 sur
la page 186 classe ainsi les données de manière optimale.

Comment un tel arbre est-il créé à partir des données ? Pour répondre à cette question nous allons à
restreignons-nous d'abord à des attributs discrets avec un nombre fini de valeurs. Parce que le
le nombre d'attributs est également fini et chaque attribut peut apparaître au plus une fois par chemin,
il existe un nombre fini d'arbres de décision différents. Un algorithme simple et évident pour
la construction d'un arbre générerait simplement tous les arbres, puis pour chaque arbre calculerait
le nombre de classifications erronées des données, et à la fin choisir l'arbre
avec le minimum d'erreurs. Ainsi nous aurions même un algorithme optimal
(au sens des erreurs pour les données d'apprentissage) pour l'apprentissage de l'arbre de décision.

L'inconvénient évident de cet algorithme est son temps de calcul trop élevé, dès que le nombre d'attributs devient
un peu plus grand. Nous allons
développer maintenant un algorithme heuristique qui, partant de la racine, construit récursivement

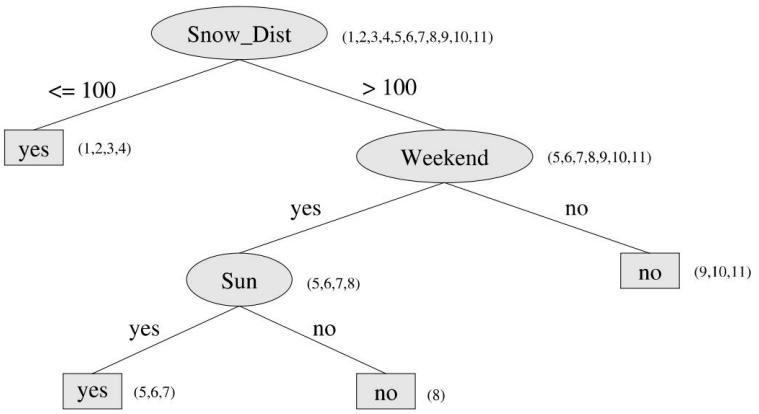


Fig. 8.23 Arbre de décision pour le problème de classification du ski. Dans les listes à droite des nœuds, les numéros des données d'apprentissage correspondantes sont indiqués. Remarquez que des nœuds feuilles ensoleillés = oui seuls deux des trois exemples sont classés correctement

Tableau 8.5 Ensemble de données pour le problème de classification du ski

Jour	Snow_Dist	Week-end	Soleil	Ski
1	≤100	Oui		Oui oui
2	≤100	Oui		Oui oui
3	≤100	Oui		Non Oui
4	≤100	Non		Oui oui
5	>100	Oui		Oui oui
6	>100	Oui		Oui oui
7	>100	Oui		Oui Non
8	>100	Oui		Non non
9	>100	Non		Oui Non
10	>100	Non		Oui Non
11	>100	Non		Non non

un arbre de décision. Tout d'abord, l'attribut avec le gain d'information le plus élevé (Snow_ Dist) est choisi pour le nœud racine dans l'ensemble de tous les attributs. Pour chaque valeur d'attribut (≤100, >100) il y a une branche dans l'arbre. Maintenant, pour chaque branche, ce processus est répété récursivement. Lors de la génération des nœuds, l'attribut avec le gain d'information le plus élevé parmi les attributs non encore utilisés est toujours choisi, dans la l'esprit d'une stratégie gourmande.

8.4.2 Entropie comme métrique pour le contenu informationnel

L'algorithme descendant décrit pour la construction d'un arbre de décision, à chaque sélectionne l'attribut avec le gain d'informations le plus élevé. Nous introduisons maintenant le

entropie comme métrique pour le contenu informationnel d'un ensemble de données d'entraînement D. Si nous ne regardons que la variable binaire ski dans l'exemple ci-dessus, alors D peut être décrit comme

$$D = (\text{oui, oui, oui, oui, oui, oui, non, non, non, non, non})$$

avec des probabilités estimées

$$p1 = P(\text{oui}) = 6/11 \text{ et } p2 = P(\text{non}) = 5/11.$$

Ici, nous avons évidemment une distribution de probabilité p = (6/11, 5/11). En général, pour un problème de classe n, cela se lit

$$p = (p1,...,pn)$$

avec

$$\sum_{je=1}^{n} pi = 1.$$

Pour introduire le contenu informationnel d'une distribution on observe deux cas extrêmes. Laissez d'abord

$$p = (1, 0, 0,..., 0). \tag{8.4}$$

Autrement dit, le premier des n événements se produira certainement et tous les autres ne se produiront pas. L'incertitude sur l'issue des événements est donc minime. En revanche, pour la distribution uniforme

$$p = \left(\frac{1}{n}, \frac{1}{n},..., \frac{1}{n}\right) \tag{8.5}$$

l'incertitude est maximale car aucun événement ne peut être distingué des autres.
Ici, Claude Shannon s'est demandé combien de bits seraient nécessaires pour encoder un tel événement. Dans le cas certain de (8.4), des bits nuls sont nécessaires car nous savons que le cas i = 1 se produit toujours. Dans le cas uniformément distribué de (8.5), il y a n possibilités équiprobables. Pour les codages binaires, log2 n bits sont nécessaires ici.
Comme toutes les probabilités individuelles sont pi = 1/n, des bits $\overline{\log 2}\frac{1}{pi}$ sont nécessaires pour ce codage.

Dans le cas général p = (p1,...,pn), si les probabilités des événements élémentaires s'écartent de la distribution uniforme, alors la valeur d'espérance H pour le nombre de bits est calculée. Pour cela, nous pondérerons toutes les valeurs $\overline{\log 2}\frac{1}{pi}$ avec leurs probabilités et obtiendrons

$$H = \sum_{je=1}^{n} pi(-\log 2\ pi) = -\sum_{je=1}^{n} pi \log 2\ pi\ .$$

Plus nous avons besoin de bits pour encoder un événement, plus l'incertitude sur le résultat est élevée. Nous définissons donc :

```
 1 P([Leuco7=0-6k]                | [Diag4=négatif] * [Age10=16-20]) = [0,132,0,156] ;
 2 P([Leuco7=6-8k]                | [Diag4=négatif] * [Age10=16-20]) = [0,257,0,281] ;
 3 P([Leuco7=8-10k] | [Diag4=negativ] * [Age10=16-20]) = [0,250,0,274] ;
 4 P([Leuco7=10-12k] | [Diag4=negativ] * [Age10=16-20]) = [0.159,0.183] ;
 5 P([Leuco7=12-15k] | [Diag4=negativ] * [Age10=16-20]) = [0.087,0.112] ;
 6 P([Leuco7=15-20k] | [Diag4=negativ] * [Age10=16-20]) = [0.032,0.056] ;
 7 P([Leuco7=20k-]                | [Diag4=négatif] * [Age10=16-20]) = [0,000,0,023] ;
 8 P([Leuco7=0-6k]                | [Diag4=négatif] * [Age10=21-25]) = [0,132,0,172] ;
 9 P([Leuco7=6-8k]                | [Diag4=négatif] * [Age10=21-25]) = [0,227,0,266] ;
10 P([Leuco7=8-10k] | [Diag4=negativ] * [Age10=21-25]) = [0,211,0,250] ;
11 P([Leuco7=10-12k] | [Diag4=negativ] * [Age10=21-25]) = [0.166,0.205] ;
12 P([Leuco7=12-15k] | [Diag4=negativ] * [Age10=21-25]) = [0.081,0.120] ;
13 P([Leuco7=15-20k] | [Diag4=negativ] * [Age10=21-25]) = [0.041,0.081] ;
14 P([Leuco7=20k-]                | [Diag4=négatif] * [Age10=21-25]) = [0,004,0,043] ;
```

Fig. 7.9 Certaines des règles LEXMED avec des intervalles de probabilité. "*" signifie " " ici

Estimation des probabilités des règles La structure du graphe de dépendances décrit la structure des règles apprises.11 Les règles ont ici des complexités différentes : il y a des règles qui ne décrivent que la distribution des diagnostics possibles
(règles a priori, par exemple (7.13)), règles qui décrivent la dépendance entre
le diagnostic et un symptôme (règles avec des conditions simples, par exemple (7.14)), et
enfin des règles décrivant la dépendance entre le diagnostic et deux symptômes, comme le montre la figure 7.9 en syntaxe PIT .

$$P(\text{Diag4} = \text{enflammé}) = 0{,}40, \quad P \tag{7.13}$$

$$(\text{Sono2} = \text{oui} \mid \text{Diag4} = \text{enflammé}) = 0{,}43, \tag{7.14}$$

$$P(\text{P4Q2} = \text{oui} \mid \text{Diag4} = \text{enflammé} \quad \text{P2Q2} = \text{oui}) = 0{,}61. \tag{7.15}$$

Pour maintenir la dépendance contextuelle des connaissances sauvegardées aussi petite que possible, toutes les règles contiennent le diagnostic dans leurs conditions et non comme des conclusions. C'est assez similaire à la construction de nombreux livres médicaux avec des formulations de la genre "Avec l'appendicite, nous voyons habituellement . . . ". Comme indiqué précédemment dans l'exemple 7.6 à la page 121, cependant, cela ne pose pas de problème car, en utilisant le bayésien formule, LEXMED met automatiquement ces règles sous la bonne forme.

Les valeurs numériques de ces règles sont estimées en comptant leur fréquence dans la base de données. Par exemple, la valeur dans (7.14) est donnée en comptant et en calculant

$$\frac{|\text{Diag4} = \text{enflammé} \quad \text{Sono2} = \text{oui}|}{|\text{Diag4} = \text{enflammé}|} = 0{,}43.$$

11La différence entre celui-ci et un réseau bayésien est, par exemple, que les règles sont équipées avec des intervalles de probabilité et que ce n'est qu'après application du principe d'entropie maximale qu'un unique modèle de probabilité produit.

Règles expertes

Étant donné que la base de données sur l'appendicite ne contient que des patients qui ont subi la opération, les règles pour les douleurs abdominales non spécifiques (NSAP) reçoivent leurs valeurs de propositions d'experts médicaux. Les expériences de LEXMED confirment que les règles probabilistes sont faciles à lire et peuvent être directement traduites en langage naturel.
Déclarations d'experts médicaux sur les relations de fréquence de symptômes spécifiques et le diagnostic, qu'il soit issu de la littérature ou à la suite d'un entretien, peut donc être incorporé dans la base de règles à peu de frais. Pour modéliser l'incertitude des connaissances d'experts, l'utilisation d'intervalles de probabilité s'est avérée efficace.
Les connaissances spécialisées ont été principalement acquises auprès des chirurgiens participants, le Dr. Rampf et Dr. Hontschik, et leurs publications [Hon94].

Une fois les règles expertes créées, la base de règles est terminée. Ensuite, le modèle de probabilité complet est calculé avec la méthode d'entropie maximale par le Système PIT.

Requêtes de diagnostic

En utilisant son modèle de probabilité efficacement stocké, LEXMED calcule les probabilités pour les quatre diagnostics possibles en quelques secondes. Par exemple, nous supposons que le sortie suivante :

	Résultats du diagnostic PIT		
Diagnostic	Appendice enflammé Appendice perforé 0,24 0,16	Négatif	Autre
Probabilité		0,57	0,03

Une décision doit être prise sur la base de ces quatre valeurs de probabilité pour poursuivre l'une des les quatre traitements : opération, opération d'urgence, observation stationnaire ou observation ambulante.12 Bien que la probabilité d'un diagnostic négatif dans ce cas l'emporte sur les autres, renvoyer le patient chez lui en bonne santé n'est pas une bonne décision.
On voit bien que, même lorsque les probabilités des diagnostics ont été calculé, le diagnostic n'est pas encore terminé.

Il s'agit plutôt maintenant de tirer une décision optimale à partir de ces probabilités. À A cette fin, l'utilisateur peut demander à LEXMED de calculer une décision recommandée.

7.3.5 Gestion des risques à l'aide de la matrice des coûts

Comment les probabilités calculées peuvent-elles désormais être traduites de manière optimale en décisions ? Un algorithme naïf attribuerait une décision à chaque diagnostic et sélectionnerait finalement la décision qui correspond à la probabilité la plus élevée. Supposons que le calcul les probabilités sont de 0,40 pour le diagnostic d'appendicite (enflammée ou perforée), 0,55 pour le diagnostic négatif et 0,05 pour le diagnostic autre. Un algorithme naïf serait choisissez maintenant la décision (trop risquée) « pas d'opération » car elle correspond au diagnostic le plus probable. Une meilleure méthode consiste à comparer les coûts

12L'observation ambulatoire signifie que le patient est libéré pour rester à domicile.

Tableau 7.3 La matrice des coûts de LEXMED avec les probabilités de diagnostic calculées d'un patient

Thérapie	Probabilité de divers diagnostics				
	Enflammé	Perforé	Négatif	Autre	
	0,25	0,15	0,55	0,05	
Opération	0	500	5800	6000	3565
Opération d'urgence	500	0	6300	6500	3915
Observation ambulante.	12000	150000	0	16500	26325
Autre	3000	5000	1300	0	2215
Observation stationnaire.	3500	7000	400	600	2175

des erreurs possibles qui peuvent survenir pour chaque décision. L'erreur est quantifiée dans le forme de « surcoût (hypothétique) de la décision actuelle par rapport à l'optimum ». Les valeurs données contiennent les coûts pour l'hôpital, pour la compagnie d'assurance, au patient (par exemple risque de complications postopératoires) et aux autres parties (par exemple absence du travail), en tenant compte des conséquences à long terme. Ces coûts sont donnés dans le tableau 7.3.

Les entrées sont finalement moyennées pour chaque décision, c'est-à-dire additionnées en prenant compte de leurs fréquences. Ceux-ci sont répertoriés dans la dernière colonne du tableau 7.3. Enfin, la décision avec le plus petit coût moyen d'erreur est suggérée. Dans le tableau 7.3 la matrice est donnée avec le vecteur de probabilité calculé pour un patient (en ce cas : (0,25, 0,15, 0,55, 0,05)). La dernière colonne du tableau contient le résultat de les calculs des coûts moyens attendus des erreurs. La valeur de l'opération dans la première ligne est donc calculé comme $0,25 \cdot 0+0,15 \cdot 500+0,55 \cdot 5800+0,05 \cdot 6000 = 3565$, une moyenne pondérée de tous les coûts. Les décisions optimales sont saisies avec des coûts (supplémentaires) de 0. Le système décide du traitement avec la moyenne minimale coût. Il s'agit donc d'un exemple d'agent orienté vers les coûts.

Matrice des coûts dans le cas binaire

Pour mieux comprendre la matrice des coûts et la gestion des risques, nous allons maintenant limiter la système LEXMED à la décision à deux valeurs entre le diagnostic d'appendicite et NSAP. Comme traitements possibles, seule opération avec probabilité p1 et ambulatoire observation avec probabilité p2 peut être choisie. La matrice de coût est donc un 2 × 2 matrice du formulaire

$$\begin{matrix} 0 & k2 \\ k1 & 0 \end{matrix} .$$

Les deux zéros dans la diagonale représentent l' opération de décision correcte dans le cas d' appendicite et observation ambulatoire pour NSAP. Le paramètre k2 représente les coûts attendus qui surviennent lorsqu'un patient sans appendice enflammé est opéré. Cette erreur est appelée un faux positif. D'autre part, la décision

l'observation ambulatoire en cas d'appendicite est un faux négatif. Le vecteur de probabilité (p1, p2) J est maintenant multiplié par cette matrice et on obtient le vecteur

$$(k2\ p2,\ k1\ p1)^{J}$$

avec le surcoût moyen des deux traitements possibles. Étant donné que la décision ne prend en compte que la relation entre les deux composants, le vecteur peut être multiplié par n'importe quel facteur scalaire. On choisit 1/k1 et on obtient ((k2/k1)p2, p1).
Ainsi seule la relation k = k2/k1 est pertinente ici. Le même résultat est obtenu par la matrice de coût plus simple

$$\begin{matrix} 0 & k \\ 1 & 0 \end{matrix},$$

qui ne contient que la variable k. Ce paramètre est très important car il conditionne la gestion des risques. En changeant k, nous pouvons adapter le "point de fonctionnement" du système de diagnostic. Pour $k \to \infty$ le système est placé dans un cadre extrêmement risqué car aucun patient ne sera jamais opéré, avec pour conséquence qu'il ne donne pas de classifications de faux positifs, mais de nombreux faux négatifs. Dans le cas où k = 0, les conditions sont exactement inverses et tous les patients sont opérés.

7.3.6 Performances

LEXMED est destiné à être utilisé dans un cabinet médical ou une ambulance. Les conditions préalables à l'utilisation de LEXMED sont des douleurs abdominales aiguës pendant plusieurs heures (mais moins de cinq jours). De plus, LEXMED est (actuellement) spécialisé pour l'appendicite, ce qui signifie que pour d'autres maladies, le système contient très peu d'informations.

Dans le cadre d'une étude prospective, une base de données représentative de 185 cas a été créée au 14 Nothelfer Hospital. Il contient les patients de l'hôpital qui sont venus à la clinique après plusieurs heures de douleurs abdominales aiguës et une suspicion d'appendicite.
A partir de ces patients, les symptômes et le diagnostic (vérifié à partir d'un prélèvement de tissu dans le cas d'une opération) sont notés.

Si les patients ont été libérés pour rentrer chez eux (sans opération) après un séjour de plusieurs heures ou 1 à 2 jours avec peu ou pas de plainte, il a ensuite été demandé par téléphone si le patient restait sans symptômes ou si un diagnostic positif avait été trouvé dans traitement ultérieur.

Pour simplifier la représentation et permettre une meilleure comparaison avec des études similaires, LEXMED a été limité à la distinction à deux valeurs entre appendicite et NSAP, comme décrit dans la Sect. 7.3.5. Maintenant, k varie entre zéro et l'infini et pour chaque valeur de k, la sensibilité et la spécificité sont mesurées par rapport aux données du test.
Mesures de sensibilité

$$P(\text{classé positif}|\text{positif}) = \frac{|\text{positif et classé positif}|}{|\text{positif}|},$$

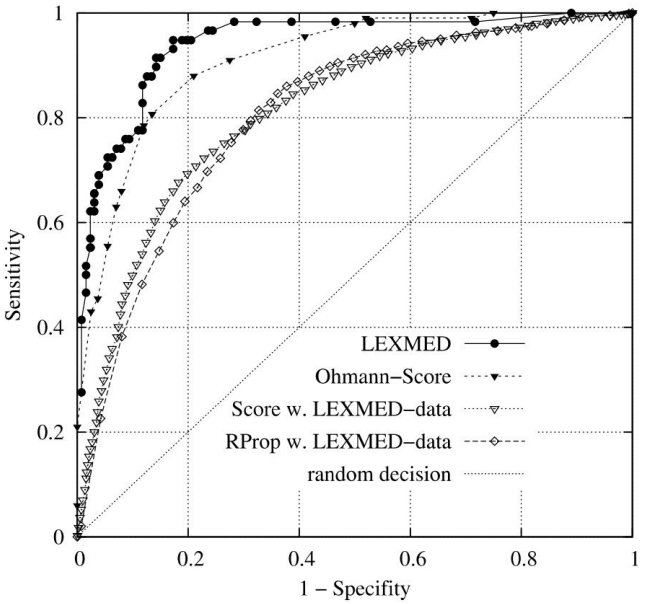


Fig. 7.10 Courbe ROC de LEXMED comparée au score d'Ohmann et à deux modèles supplémentaires

c'est-à-dire la part relative des cas positifs qui sont correctement identifiés. Il indique la sensibilité du système de diagnostic. La spécificité, en revanche, mesure

$$P(\text{classé négatif}|\text{négatif}) = |\text{négatif}| \quad \frac{|\text{négatif et classé négatif}|}{\qquad\qquad\qquad\qquad},$$

c'est-à-dire la part relative des cas négatifs qui sont correctement identifiés.

Nous donnons les résultats de la sensibilité et de la spécificité dans la Fig. 7.10 pour $0 \le k < \infty$. Cette courbe est notée courbe ROC, ou caractéristique de fonctionnement du récepteur. Avant d'en venir à l'analyse de la qualité de LEXMED, quelques mots sur la signification de la courbe ROC. La ligne bissectrice du diagramme en diagonale est tracée pour l'orientation.
Tous les points sur cette ligne correspondent à une décision aléatoire. Par exemple, le point (0,2, 0,2) correspond à une spécificité de 0,8 avec une sensibilité de 0,2. On peut y arriver assez facilement en classant un nouveau cas, sans le regarder, avec des probabilités de 0,2 pour le positif et de 0,8 pour le négatif. Chaque système de diagnostic basé sur la connaissance doit donc générer un ROC qui se situe clairement au-dessus de la diagonale.

Les valeurs extrêmes de la courbe ROC sont également intéressantes. Au point (0, 0) les trois courbes se croisent. Le système de diagnostic correspondant classerait tous les cas comme négatifs. L'autre valeur extrême (1, 1) correspond à un système qui déciderait de faire l'opération pour chaque patient et a donc une sensibilité de 1. On pourrait appeler la courbe ROC la courbe caractéristique des systèmes de diagnostic à deux valeurs. Le système de diagnostic idéal aurait une courbe caractéristique constituée uniquement du point (0, 1) et aurait donc une spécificité de 100 % et une sensibilité de 100 %.

Analysons maintenant la courbe ROC. À une sensibilité de 88 %, LEXMED atteint une spécificité de 87 % (k = 0,6). A titre de comparaison, le score d'Ohmann, un score établi et bien connu pour l'appendicite est donné [OMYL96, ZSR+99]. Parce que LEXMED est au-dessus ou à gauche du score d'Ohmann presque partout, sa qualité moyenne de diagnostic est nettement meilleure. Ce n'est pas surprenant car les scores sont tout simplement trop faibles pour modéliser des propositions intéressantes. Insecte. 8.6 et dans l'exercice 8.16 à la page 219, nous montrerons que les scores sont équivalents au cas particulier de Bayes naïf, c'est-à-dire à l'hypothèse que tous les symptômes sont deux à deux indépendants lorsque le diagnostic est connu. Lors de la comparaison de LEXMED avec des scores, il convient cependant de mentionner qu'une base de données statistiquement représentative a été utilisée pour le score d'Ohmann, mais qu'une base de données non représentative enrichie de connaissances d'experts a été utilisée pour LEXMED .

Pour avoir une idée de la qualité des données LEXMED par rapport aux données d'Ohmann, un score linéaire a été calculé en utilisant la méthode des moindres carrés (voir section 9.4.1), qui est également tirée à titre de comparaison. De plus, un réseau de neurones a été entraîné sur les données LEXMED avec l'algorithme RProp (voir section 9.5). La force de la combinaison des données et des connaissances d'experts apparaît clairement dans la différence entre la courbe LEXMED et les courbes du système de score et de l'algorithme RProp.

7.3.7 Domaines d'application et expériences

LEXMED ne doit pas remplacer le jugement d'un chirurgien expérimenté. Cependant, parce qu'un spécialiste n'est pas toujours disponible dans un contexte clinique, une requête LEXMED offre une deuxième opinion substantielle. L'application du système dans une ambulance clinique et pour les médecins généralistes est particulièrement intéressante et utile.

La capacité d'apprentissage de LEXMED, qui permet de prendre en compte d'autres symptômes, d'autres données du patient et d'autres règles, offre également de nouvelles possibilités en clinique. Pour les groupes particulièrement rares et difficiles à diagnostiquer, par exemple les enfants de moins de six ans, LEXMED peut utiliser les données des pédiatres ou d'autres bases de données spéciales, pour soutenir même les chirurgiens expérimentés.

Outre l'utilisation directe dans le diagnostic, LEXMED prend également en charge les mesures d'assurance qualité. Par exemple, les compagnies d'assurance peuvent comparer la qualité du diagnostic des hôpitaux à celle des systèmes experts. En développant davantage la matrice des coûts créée dans LEXMED (avec le consentement des médecins, des assurances et des patients), la qualité des diagnostics des médecins, des diagnostics informatiques et des autres institutions médicales deviendra plus facile à comparer.

LEXMED a mis en évidence une nouvelle façon de construire des systèmes de diagnostic automatiques. En utilisant le langage de la théorie des probabilités et l'algorithme MaxEnt, les connaissances dérivées de manière inductive et statistique sont combinées avec les connaissances des experts et de la littérature. L'approche basée sur des modèles probabilistes est théoriquement élégante, généralement applicable et a donné de très bons résultats dans une petite étude.

LEXMED est utilisé dans la pratique à l'hôpital 14 Nothelfer à Weingarten depuis 1999 et y a très bien fonctionné. Il est également disponible sur www.lexmed.de, sans garantie, bien sûr. Sa qualité de diagnostic est comparable à celle d'un chirurgien expérimenté et est donc meilleure que celle d'un généraliste moyen, ou celle d'un médecin inexpérimenté en clinique.

Malgré ce succès, il est devenu évident qu'il est très difficile de commercialiser un tel système dans le système médical allemand. L'une des raisons en est qu'il n'existe pas de marché libre pour promouvoir une meilleure qualité (ici de meilleurs diagnostics) à travers ses mécanismes de sélection. De plus, en médecine, le temps d'une large utilisation des techniques intelligentes n'est pas encore venu, même en 2010. Une des causes pourrait être les enseignements conservateurs à cet égard dans les facultés de médecine allemandes.

Un autre problème est le désir de nombreux patients d'obtenir des conseils et des soins personnels de la part du médecin, ainsi que la crainte qu'avec l'introduction de systèmes experts, le patient ne communique qu'avec la machine. Cette crainte, cependant, est totalement infondée. Même à long terme, les systèmes experts médicaux ne peuvent remplacer le médecin. Ils peuvent cependant, tout comme la chirurgie au laser et l'imagerie par résonance magnétique, être utilisés avantageusement pour tous les participants. Depuis le premier système de diagnostic informatique médical de de Dombal en 1972, près de 40 ans se sont écoulés. Il reste à voir si la médecine attendra encore 40 ans avant que le diagnostic informatique ne devienne un outil médical établi.

7.4 Raisonnement avec les réseaux bayésiens

Un problème avec le raisonnement utilisant la probabilité dans la pratique a déjà été souligné dans la Sect. 7.1. Si d variables X1,...,Xd avec n valeurs chacune sont utilisées, alors les d valeurs la distribution de probabilité utilise totales associées. Cela signifie que dans le pire des cas, le n mémoire et le temps de calcul pour déterminer les probabilités spécifiées croît de manière exponentielle avec le nombre de variables.

En pratique, les applications sont généralement très structurées et la distribution comporte de nombreuses redondances. Cela signifie qu'il peut être fortement réduit avec les méthodes appropriées. L'utilisation de réseaux bayésiens a ici prouvé sa puissance et fait partie des techniques d'IA qui ont été utilisées avec succès dans la pratique. Les réseaux bayésiens utilisent les connaissances sur l'indépendance des variables pour simplifier le modèle.

7.4.1 Variables indépendantes

Dans le cas le plus simple, toutes les variables sont indépendantes deux à deux et c'est le cas que

$$P(X1,...,Xd) = P(X1) \cdot P(X2) \cdots P(Xd).$$

Toutes les entrées de la distribution peuvent ainsi être calculées à partir des d valeurs P (X1), . . . , P (Xd). Cependant, les applications intéressantes ne peuvent généralement pas être modélisées car les probabilités conditionnelles deviennent triviales.13

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

13Dans la méthode naïve de Bayes, l'indépendance de tous les attributs est supposée, et cette méthode a été appliquée avec succès à la classification de textes (voir section 8.6).

toutes les probabilités conditionnelles sont réduites aux probabilités a priori. La situation devient plus intéressante lorsqu'une partie seulement des variables sont indépendantes ou dépendantes sous certaines conditions. Pour raisonner en IA, les dépendances entre les variables sont importantes et doivent être utilisées.

Nous aimerions esquisser le raisonnement avec les réseaux bayésiens à travers un exemple simple et très illustratif de J. Pearl [Pea88], devenu bien connu grâce à [RN10] et qui est maintenant une connaissance de base de l'IA.

Exemple 7.7 (Exemple d'alarme) Bob, qui est célibataire, a fait installer un système d'alarme dans sa maison pour se protéger contre les cambrioleurs. Bob ne peut pas entendre l'alarme lorsqu'il travaille au bureau. C'est pourquoi il a demandé à ses deux voisins, John dans la maison voisine à gauche, et Mary dans la maison à droite, de l'appeler à son bureau s'ils entendent son alarme. Après quelques années, Bob sait à quel point John et Mary sont fiables et modélise leur comportement d'appel en utilisant la probabilité conditionnelle comme suit.14

$$P(J|Al) = 0{,}90 \; P(M|Al) = 0{,}70, \; P(J|\neg Al) = 0{,}05 \; P(M|\neg Al) = 0{,}01.$$

Comme Marie est malentendante, elle n'entend pas l'alarme plus souvent que Jean. Cependant, John confond parfois l'alarme de la maison de Bob avec les alarmes d'autres maisons. L'alarme est déclenchée par un cambriolage, mais peut également être déclenchée par un (faible) tremblement de terre, ce qui peut entraîner une fausse alarme car Bob ne veut être informé des cambriolages que lorsqu'il est à son bureau. Ces relations sont modélisées par

$$P(Al|Frais, Oreille) = 0{,}95,$$

$$P(Al|Frais, \neg Ear) = 0{,}94,$$

$$P(Al|\neg Four, Oreille) = 0{,}29,$$

$$P(Al|\neg Four, \neg Oreille) = 0{,}001,$$

ainsi que les probabilités a priori P (Bur) = 0,001 et P (Ear) = 0,002. Ces deux variables sont indépendantes car les séismes ne se planifient pas en fonction des habitudes des cambrioleurs, et à l'inverse il n'existe aucun moyen de prévoir les séismes, les cambrioleurs n'ont donc pas la possibilité de fixer leur horaire en conséquence.

Des requêtes sont désormais effectuées sur cette base de connaissances. Par exemple, Bob pourrait être intéressé par P (Bur|J M), P (J |Bur) ou P (M|Bur). C'est-à-dire qu'il veut savoir avec quelle sensibilité les variables J et M réagissent à un rapport de cambriolage.

14Les variables binaires J et M représentent les deux événements « Jean appelle » et « Marie appelle », respectivement Al pour « sirène d'alarme retentit », Bur pour « cambriolage » et Ear pour « tremblement de terre ».

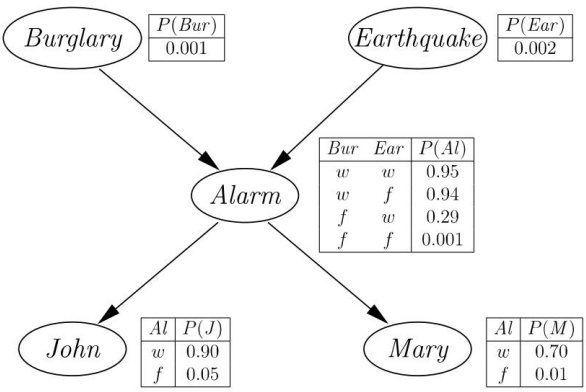


Fig. 7.11 Réseau bayésien pour l'exemple d'alarme avec les CPT associés

7.4.2 Représentation graphique des connaissances sous forme de réseau bayésien

Nous pouvons grandement simplifier le travail pratique en représentant graphiquement les connaissances formulées sous forme de probabilité conditionnelle. La figure 7.11 montre le réseau bayésien pour l'exemple d'alarme. Chaque nœud du réseau représente une variable et chaque arête dirigée une déclaration de probabilité conditionnelle. L'arête de Al à J représente par exemple les deux valeurs P (J |Al) et P (J |¬Al), qui est donnée sous la forme d'un tableau, dit CPT (table de probabilité conditionnelle). Le CPT d'un nœud liste toutes les probabilités conditionnelles de la variable du nœud conditionnées sur tous les nœuds connectés par des arêtes entrantes.

En étudiant le réseau, nous pourrions nous demander pourquoi il n'y a pas d'autres arêtes incluses en plus des quatre qui sont dessinées. Les deux nœuds Bur et Ear ne sont pas liés puisque les variables sont indépendantes. Tous les autres nœuds ont un nœud parent, ce qui rend le raisonnement un peu plus complexe. Nous avons d'abord besoin du concept d'indépendance conditionnelle.

7.4.3 Indépendance conditionnelle

De manière analogue à l'indépendance des variables aléatoires, nous donnons

Définition 7.6 Deux variables A et B sont dites conditionnellement indépendantes, étant donné C si

$$P(A, B|C) = P(A|C) \cdot P(B|C).$$

Cette équation est vraie pour toutes les combinaisons de valeurs pour les trois variables (c'est-à-dire pour la distribution), que nous voyons dans la notation. Examinons maintenant les nœuds J et M dans l'exemple d'alarme, qui ont le nœud parent commun Al. Si Jean et Marie

réagissent indépendamment à une alarme, alors les deux variables J et M sont indépendantes étant donné Al, soit :

$$P(J,M|Al) = P(J|Al) \cdot P(M|Al).$$

Si la valeur de Al est connue, par exemple parce qu'une alarme s'est déclenchée, alors les variables J et M sont indépendantes (sous la condition Al = w). Du fait de l'indépendance conditionnelle des deux variables J et M, aucune arête entre ces deux nœuds n'est ajoutée. Cependant, J et M ne sont pas indépendants (voir Exercice 7.11 à la page 159).

La relation entre les deux variables J et Bur est assez similaire, car John ne réagit pas à un cambriolage, mais plutôt à l'alarme. Cela pourrait être, par exemple, à cause d'un haut mur qui bloque sa vue sur la propriété de Bob, mais il peut toujours entendre l'alarme. Ainsi J et Bur sont indépendants étant donné Al et

$$P(J,Bur|Al) = P(J|Al) \cdot P(Bur|Al).$$

Étant donné une alarme, les variables J et Ear, M et Bur, ainsi que M et Ear sont également indépendantes. Pour le calcul avec indépendance conditionnelle, les caractérisations suivantes, qui sont équivalentes à la définition ci-dessus, sont utiles :

Théorème 7.5 Les équations suivantes sont deux à deux équivalentes, ce qui signifie que chaque équation individuelle décrit l'indépendance conditionnelle des variables A et B sachant C.

$$P(A, B|C) = P(A|C) \cdot P(B|C), \quad (7.16)$$

$$P(A|B,C) = P(A|C), \quad (7.17)$$

$$P(B|A,C) = P(B|C). \quad (7.18)$$

Preuve D'une part, en utilisant l'indépendance conditionnelle (7.16), nous pouvons conclure que

$$P(A, B, C) = P(A, B|C)P(C) = P(A|C)P(B|C)P(C).$$

D'autre part, la règle du produit nous donne

$$P(A, B, C) = P(A|B,C)P(B|C)P(C).$$

Ainsi P(A|B,C) = P(A|C) est équivalent à (7.16). On obtient (7.18) de manière analogue en intervertissant A et B dans cette dérivation.

7.4.4 Application pratique

Revenons maintenant à l'exemple d'alarme et montrons comment le réseau bayésien de la Fig. 7.11 à la page 146 peut être utilisé pour le raisonnement. Bob s'intéresse, par exemple, à la sensibilité de ses deux reporters d'alarme John et Mary, c'est-à-dire à P (J |Bur) et P (M| Bur). Cependant, les valeurs P (Bur|J) et P (Bur|M), ainsi que P (Bur|J,M) sont encore plus importantes pour lui. On commence par P (J |Bur) et on calcule

$$P(J \mid Bur) = \frac{P(J, Bur)}{P(fraise)} = \frac{P(J,Bur,Al) + P(J,Bur,\neg Al)}{P(fraise)} \tag{7.19}$$

et

$$P(J,Four,Al) = P(J \mid Four,Al)P(Al|Four)P(Four) = P(J \mid Al)P(Al|Four)P(Four), \tag{7.20}$$

où pour les deux dernières équations nous avons utilisé la règle du produit et l'indépendance conditionnelle de J et Bur étant donné Al. Inséré dans (7.19) on obtient

$$\begin{aligned} P(J \mid Fraise) &= \frac{P(J \mid Al)P(Al|Bur)P(Bur) + P(J \mid \neg Al)P(\neg Al|Bur)P(Bur)}{P(Fraise)} \\ &= P(J \mid Al)P(Al|Bur) + P(J \mid \neg Al)P(\neg Al|Bur). \end{aligned} \tag{7.21}$$

Ici P (Al|Bur) et P (¬Al|Bur) manquent. On calcule donc

$$\begin{aligned} P(Al|Fur) &= \frac{P(Al, Bur)}{P(fraise)} = \frac{P(Al, Bur, Oreille) + P(Al, Bur, \neg\ Oreille)}{P(fraise)} \\ &= \frac{P(Al|Frais,Ear)P(Frais)P(Ear) + P(Al|Fur,\neg Ear)P(Frais)P(\neg Ear)}{P(fraise)} \\ &= P(Al|Frais,Ear)P(Ear) + P(Al|Fur,\neg Ear)P(\neg Ear) \\ &= 0{,}95 \cdot 0{,}002 + 0{,}94 \cdot 0{,}998 = 0{,}94 \end{aligned}$$

ainsi que P (¬Al|Bur) = 0,06 et l'insérer dans (7.21) ce qui donne le résultat

$$P(J \mid Bur) = 0{,}9 \cdot 0{,}94 + 0{,}05 \cdot 0{,}06 = 0{,}849.$$

De manière analogue, nous calculons P (M|Bur) = 0,659. Nous savons maintenant que John appelle environ 85 % de tous les cambriolages et Mary environ 66 % de tous les cambriolages. La probabilité que les deux appellent est calculée, en raison de l'indépendance conditionnelle, comme

$$P(J,M|Four) = P(J \mid Four)P(M|Four) = 0{,}849 \cdot 0{,}659 = 0{,}559.$$

Plus intéressant, cependant, est la probabilité d'un appel de John ou Mary

$$P(J \quad M|Bur) = P(\neg(\neg J,\neg M)|Bur) = 1 - P(\neg J,\neg M|Bur)$$
$$= 1 - P(\neg J|Bur)P(\neg M|Bur) = 1 - 0{,}051 = 0{,}948.$$

Bob reçoit ainsi une notification pour environ 95 % de tous les cambriolages. Maintenant pour calculer P (Bur|J), on applique la formule de Bayes, qui nous donne

$$P(\text{Four}|J) = \frac{P(J|\text{Fraise})P(\text{Fraise})}{P\,J)} = \frac{0{,}849 \cdot 0{,}001}{0{,}052} = 0{,}016.$$

De toute évidence, seulement 1,6 % environ de tous les appels de John sont en fait dus à un cambriolage. Étant donné que la probabilité de fausses alarmes est cinq fois plus faible pour Mary, avec P (Bur|M) = 0,056, nous avons une confiance significativement plus élevée compte tenu d'un appel de Mary. Bob ne devrait s'inquiéter sérieusement de sa maison que si les deux appellent, car P (Bur|J,M) = 0,284 (voir l'exercice 7.11 à la page 159).

Dans (7.21) à la page 148 nous avons montré avec

$$P(J|Bur) = P(J|Al)P(Al|Bur) + P(J|\neg Al)P(\neg Al|Bur)$$

comment nous pouvons "insérer" une nouvelle variable. Cette relation vaut en général pour deux variables A et B compte tenu de l'introduction d'une variable supplémentaire C et est appelée conditionnement :

$$P(A|B) = \sum_c P(A|B,C = c)P(C = c|B).$$

Si de plus A et B sont conditionnellement indépendants étant donné C, cette formule se simplifie en

$$P(A|B) = \sum_c P(A|C = c)P(C = c|B).$$

7.4.5 Logiciel pour réseaux bayésiens

Nous donnerons une brève introduction à deux outils en utilisant l'exemple d'alarme. Nous connaissons déjà le système PIT. Nous saisissons les valeurs des CPT dans la syntaxe PIT dans la fenêtre de saisie en ligne sur www.pit-systems.de. Après l'entrée illustrée à la Fig. 7.12 à la page 150 , nous recevons la réponse :

P([Einbruch=t] | [Jean=t] ET [Marie=t]) = 0,2841.

Bien que PIT ne soit pas un outil de réseau bayésien classique, il peut prendre des probabilités conditionnelles arbitraires et des requêtes en entrée et calculer des résultats corrects. On peut montrer [Sch96], qu'en entrée de CPT ou de règles équivalentes, le principe MaxEnt implique les mêmes indépendances conditionnelles et donc aussi les mêmes réponses qu'un réseau bayésien. Les réseaux bayésiens sont donc un cas particulier de MaxEnt.

```
1 var Alarme{t,f}, Cambriolage{t,f}, Tremblement de terre{t,f}, Jean{t,f}, Marie{t,f} ;
2
3 P([Séisme=t]) = 0,002 ; 4 P([Cambriolage=t])
= 0,001 ; 5 P([Alarme=t] | [Cambriolage=t]
ET [Séisme=t]) = 0,95 ; 6 P([Alarme=t] | [Cambriolage=t] ET [Séisme=f]) = 0,94 ; 7
P([Alarme=t] | [Cambriolage=f] ET [Séisme=t]) = 0,29 ; 8 P([Alarme=t] | [Cambriolage=f] ET
[Séisme=f]) = 0,001 ; 9 P([Jean=t] | [Alarme=t]) = 0,90 ; 10 P([Jean=t] | [Alarme=f]) = 0,05 ;
11 P([Marie=t] | [Alarme=t]) = 0,70 ; 12 P([Marie=t] | [Alarme=f]) = 0,01 ; 13 14
QP([Cambriolage=t] | [Jean=t] AND [Marie=t]);
```

Fig. 7.12 Entrée PIT pour l'exemple d'alarme

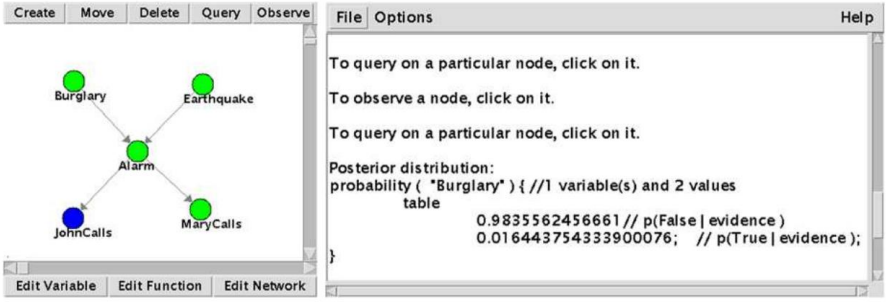


Fig. 7.13 L'interface utilisateur de JavaBayes : à gauche l'éditeur graphique et à droite la console où les réponses sont données en sortie

Ensuite, nous examinerons JavaBayes [Coz98], un système classique également disponible gratuitement sur Internet avec l'interface graphique illustrée à la Fig. 7.13. Avec l'éditeur de réseau graphique, les nœuds et les bords peuvent être manipulés et les valeurs dans les CPT éditées. De plus, les valeurs des variables peuvent être assignées avec « Observer » et les valeurs des autres variables appelées avec « Query ». Les réponses aux requêtes apparaissent alors dans la fenêtre de la console.

Le système professionnel et commercial Hugin est nettement plus puissant et pratique. Par exemple, Hugin peut utiliser des variables continues en plus des variables discrètes. Il peut également apprendre les réseaux bayésiens, c'est-à-dire générer le réseau de manière entièrement automatique à partir de données statistiques (voir section 8.5).

7.4.6 Développement de réseaux bayésiens

Un réseau bayésien compact est très clair et significativement plus informatif pour le lecteur qu'une distribution de probabilité complète. De plus, il nécessite beaucoup moins

mémoire. Pour les variables v1,...,vn avec |v1|,...,|vn| différentes valeurs chacune, la distribution a un total de

$$\prod_{je=1}^{n} |vi| - 1$$

entrées indépendantes. Dans l'exemple d'alarme, les variables sont toutes binaires. Ainsi pour toutes les variables |vi | = 2, et la distribution a 25 − 1 = 31 entrées indépendantes. Pour calculer le nombre d'entrées indépendantes pour le réseau bayésien, le nombre total de toutes les entrées de tous les CPT doit être déterminé. Pour un nœud vi avec ki nœuds parents ei1,...,eiki , le CPT associé a

$$(|v| - 1) \prod_{j=1}^{ki} |eij|$$

entrées. Ensuite, tous les CPT du réseau ont ensemble

$$\prod_{je=1}^{n} (|v| - 1) \prod_{j=1}^{ki} |eij| \qquad (7.22)$$

entrées.15 Pour l'exemple d'alarme, le résultat est alors

$$2 + 2 + 4 + 1 + 1 = 10$$

des entrées indépendantes qui décrivent de manière unique le réseau. La comparaison de la complexité en mémoire entre la distribution complète et le réseau bayésien devient plus claire lorsque nous supposons que toutes les n variables ont le même nombre b de valeurs et que chaque nœud a k nœuds parents. Alors (7.22) peut être simplifié et tous les CPT ont ensemble n(b − 1)bk entrées. La distribution complète contient $b^n - 1$ entrées. Une signification cant gain n'est faite alors que si le nombre moyen de nœuds parents est beaucoup plus petit que le nombre de variables. Cela signifie que les nœuds ne sont connectés que localement. En raison de la connexion locale, le réseau devient modularisé, ce qui, comme dans le génie logiciel, conduit à une réduction de la complexité. Dans l'exemple d'alarme, le nœud d'alarme sépare les nœuds Bur et Ear des nœuds J et M. Nous pouvons également le voir clairement dans l' exemple LEXMED .

15Pour le cas d'un nœud sans ancêtres le produit dans cette somme est vide. Pour cela, nous substituons la valeur 1 car le CPT pour les nœuds sans ancêtres contient, avec sa probabilité a priori, exactement une valeur.

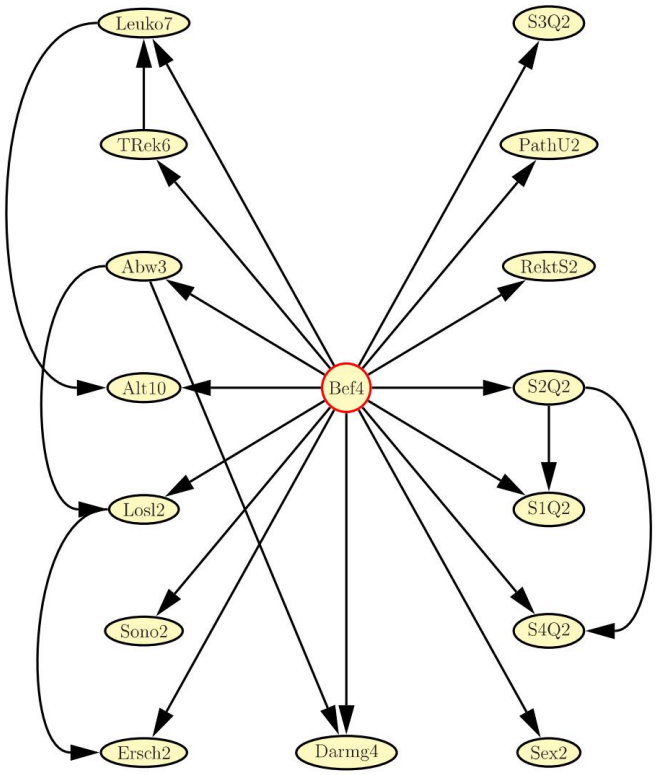


Fig. 7.14 Réseau bayésien pour l' application LEXMED

LEXMED en tant que réseau bayésien

Le système LEXMED décrit dans la Sect. 7.3 peut également être modélisé comme un réseau bayésien. En orientant les lignes extérieures finement dessinées (en leur donnant des flèches), le graphique d'indépendance de la Fig. 7.8 à la page 137 peut être interprété comme un réseau bayésien. Le réseau résultant est illustré à la Fig. 7.14.

Insecte. 7.3.2 la taille de la distribution pour LEXMED a été calculée comme la valeur 20 643 839. Le réseau bayésien, en revanche, peut être entièrement décrit avec seulement 521 valeurs. Cette valeur peut être déterminée en entrant les variables de la Fig. 7.14 dans (7.22) à la page 151. Dans l'ordre (Leuko, TRek, Gua, Age, Reb, Sono, Tapp, BowS, Sex, P4Q, P1Q, P2Q, RecP , Urin, P3Q, Diag4) nous calculons

$$\sum_{je=1}^{n} (|v| - 1) \quad \sum_{ki} |eij| = 6 \cdot 6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 7 \cdot 4 + 1 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 2 \cdot 4 \; j=1$$

$$+ 3 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 4$$

$$+ 1 \cdot 4 + 1 = 521.$$

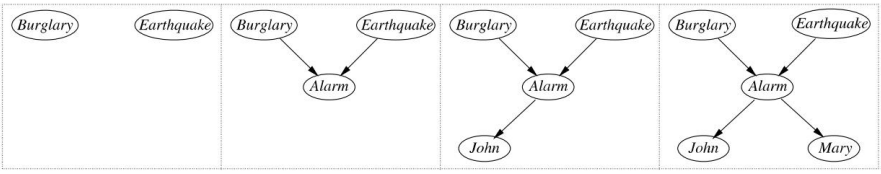


Fig. 7.15 Construction par étapes du réseau d'alarme en tenant compte de la causalité

Cet exemple démontre qu'il est pratiquement impossible de construire une distribution complète pour des applications réelles. Un réseau bayésien avec 22 arêtes et 521 valeurs de probabilité, en revanche, est toujours gérable.

Causalité et structure du réseau La
construction d'un réseau bayésien se déroule généralement en deux étapes.

1. Conception de la structure du réseau : Cette étape est généralement réalisée manuellement et sera décrite ci-après.
2. Saisie des probabilités dans les CPT : La saisie manuelle des valeurs dans le cas de nombreuses variables est très fastidieuse. Si (comme par exemple avec LEXMED) une base de données est disponible, cette étape peut être automatisée en estimant les entrées CPT en comptant les fréquences.

Nous allons maintenant décrire la construction du réseau dans l'exemple d'alarme (voir Fig. 7.15). Au début, nous connaissons les deux causes Cambriolage et Tremblement de terre et les deux symptômes John et Mary. Cependant, comme John et Mary ne réagissent pas directement à un cambrioleur ou à un tremblement de terre, mais uniquement à l'alarme, il convient d'ajouter ceci comme une variable supplémentaire qui n'est pas observable par Bob. Le processus d'ajout d'arcs commence par les causes, c'est-à-dire par les variables qui n'ont pas de nœud parent. D'abord, nous choisissons Cambriolage et ensuite Tremblement de terre. Maintenant, nous devons vérifier si Tremblement de terre est indépendant de Cambriolage. Ceci est donné, et donc aucun avantage n'est ajouté du cambriolage au tremblement de terre. Étant donné que l'alarme dépend directement du cambriolage et du tremblement de terre, ces variables sont choisies ensuite et un front est ajouté à la fois du cambriolage et du tremblement de terre à l'alarme. Ensuite, nous choisissons John. Étant donné que Alarm et John ne sont pas indépendants, un front est ajouté de l'alarme à John. Il en est de même pour Marie. Maintenant, nous devons vérifier si John est conditionnellement indépendant de l'alarme de cambriolage donnée . Si ce n'est pas le cas, une autre arête doit être insérée de Cambriolage à Jean. Nous devons également vérifier si des arêtes sont nécessaires du tremblement de terre à John et du cambriolage ou du tremblement de terre à Mary. En raison de l'indépendance conditionnelle, ces quatre arêtes ne sont pas nécessaires. Les arêtes entre John et Mary sont également inutiles car John et Mary sont conditionnellement indépendants étant donné Alarm.

La structure du réseau bayésien dépend fortement de l'ordre des variables choisi. Si l'ordre des variables est choisi pour refléter la relation causale en commençant par les causes et en passant aux variables de diagnostic, alors le résultat sera un réseau simple. Sinon, le réseau peut contenir beaucoup plus de tronçons.
De tels réseaux non causals sont souvent très difficiles à comprendre et ont une plus grande

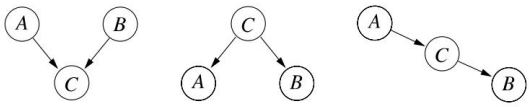


Fig. 7.16 Il n'y a pas d'arête entre A et B s'ils sont indépendants (gauche) ou conditionnellement indépendants (milieu, droite)

complexité du raisonnement. Le lecteur peut se référer à l'exercice 7.11 à la page 159 pour une meilleure compréhension.

7.4.7 Sémantique des réseaux bayésiens

Comme nous l'avons vu dans la section précédente, aucune arête n'est ajoutée à un réseau bayésien entre deux variables A et B lorsque A et B sont indépendants ou conditionnellement indépendants étant donné une troisième variable C. Cette situation est représentée sur la figure 7.16 .

Nous exigeons maintenant que le réseau bayésien n'ait pas de cycles et nous supposons que les variables sont numérotées de telle sorte qu'aucune variable n'ait un numéro inférieur à toute variable qui la précède. Ceci est toujours possible lorsque le réseau n'a pas de cycles.16 Ensuite, en utilisant toutes les indépendances conditionnelles, nous avons

$$P(Xn|X1,...,Xn-1) = P(Xn|Parents(Xn)).$$

Cette équation est donc une proposition selon laquelle une variable arbitraire Xi dans un réseau bayésien est conditionnellement indépendante de ses ancêtres, étant donné ses parents. La proposition un peu plus générale illustrée à la Fig. 7.17 à la page 155 peut être énoncée de manière compacte

comme

> Théorème 7.6 Un nœud dans un réseau bayésien est conditionnellement indépendant de tous les nœuds non successeurs, étant donné ses parents.

Nous sommes maintenant en mesure de simplifier considérablement la règle de la chaîne ((7.1) à la page 120) :

$$P(X1,...,Xn) = \prod_{je=1}^{n} P(Xi|X1 ...,Xi-1) = \prod_{je=1}^{n} P(Xi \mid Parents(Xi)). \quad (7.23)$$

En utilisant cette règle, nous pourrions, par exemple, écrire (7.20) à la page 148 directement

$$P(J,Bur,Al) = P(J|Al)P(Al|Bur)P(Bur).$$

16Si par exemple trois nœuds X1, X2, X3 forment un cycle, alors il y a les arêtes (X1, X2), (X2, X3) et (X3, X1) où X1 a X3 comme successeur.

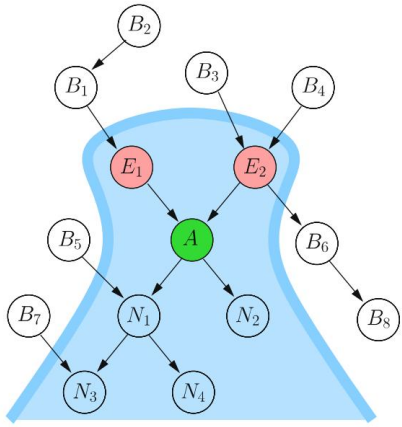


Fig. 7.17 Exemple d'indépendance conditionnelle dans un réseau bayésien. Si les nœuds parents E1 et E2 sont donnés, alors tous les nœuds non successeurs B1,...,B8 sont indépendants de A

Nous connaissons maintenant les concepts et les fondements les plus importants du réseau bayésien travaux. Résumons-les [Jen01] :

Définition 7.7 Un réseau bayésien est défini par : • Un ensemble de variables et un ensemble d'arêtes orientées entre ces variables. • Chaque variable a un nombre fini de valeurs possibles. • Les variables ainsi que les arêtes forment un graphe orienté acyclique (DAG).

Un DAG est un graphe sans cycles, c'est-à-dire sans chemins de la forme (A, . . . , UN).

• Pour chaque variable A, le CPT (c'est-à-dire le tableau des probabilités conditionnelles P(A|Parents(A))) est donné.

Deux variables A et B sont dites conditionnellement indépendantes sachant C si P(A, B|C) = P(A| C) · P(B|C) ou, de façon équivalente, si P(A|B,C) = P(A|C).

Outre les règles fondamentales de calcul des probabilités, les règles suivantes sont également vraies : P (B|A)·P (A)

Théorème de Bayes : P (A|B) = P (B) ————————

Marginalisation : P (B) = P (A, B) + P (¬A,B) = P (B|A) · P (A) + P (B|¬A) · P (¬A)

Conditionnement : P (A|B) = P (A|B,C_c= c)P (C = c|B)

Une variable dans un réseau bayésien est conditionnellement indépendante de toutes les variables non successeurs compte tenu de ses variables parentes. Si X1,...,Xn−1 ne succèdent pas à Xn, on a P(Xn|X1,...,Xn−1) = P(Xn|Parents(Xn)). Cette condition doit être respectée lors de la construction d'un réseau.

Lors de la construction d'un réseau bayésien, les variables doivent être ordonnées en fonction de la causalité. D'abord les causes, puis les variables cachées et enfin les variables de diagnostic.

Règle de chaîne : P(X1,...,Xn) = $\prod_{ni=1}$ P(Xi | Parents(Xi))

Dans [Pea88] et [Jen01], le terme d-séparation est introduit pour les réseaux bayésiens, d'où découle un théorème similaire au théorème 7.6 page 154 . Nous nous abstiendrons d'introduire ce terme et nous parviendrons ainsi à une représentation un peu plus simple, bien que théoriquement moins nette.

7.5 Résumé

D'une manière qui reflète la tendance prolongée et soutenue vers les systèmes probabilistes, nous avons introduit une logique probabiliste pour raisonner avec des connaissances incertaines. Après avoir introduit le langage - logique propositionnelle augmentée de probabilités ou d'intervalles de probabilité - nous avons choisi l'approche naturelle, bien qu'inhabituelle, via la méthode d'entropie maximale comme point d'entrée et montré comment nous pouvons modéliser un raisonnement non monotone avec cette méthode. Les réseaux bayésiens ont ensuite été introduits comme un cas particulier de la méthode MaxEnt.

Pourquoi les réseaux bayésiens sont-ils un cas particulier de MaxEnt ? Lors de la construction d'un réseau bayésien, des hypothèses sur l'indépendance sont faites qui ne sont pas nécessaires pour la méthode MaxEnt. De plus, lors de la construction d'un réseau bayésien, tous les CPT doivent être complètement remplis afin qu'une distribution de probabilité complète puisse être construite. Sinon, le raisonnement est restreint ou impossible. Avec MaxEnt, en revanche, le développeur peut formuler toutes les connaissances dont il dispose sous forme de probabilités. MaxEnt complète ensuite le modèle et génère la distribution. Même si (par exemple lors d'un entretien avec un expert) seules des propositions vagues sont disponibles, cela peut être modélisé de manière appropriée. Une proposition telle que "Je suis presque sûr que A est vrai". peut par exemple être modélisée en utilisant $P(A) \subseteq [0{,}6, 1]$ comme intervalle de probabilité. Lors de la construction d'un réseau bayésien, une valeur concrète doit être donnée pour la probabilité, si nécessaire par estimation. Cela signifie cependant que l'expert ou le développeur insère des informations ad hoc dans le système. Un autre avantage de MaxEnt est la possibilité de formuler des propositions (presque) arbitraires. Pour les réseaux bayésiens, les CPT doivent être renseignés.

La liberté dont dispose le développeur lors de la modélisation avec MaxEnt peut être un inconvénient (surtout pour un débutant) car, contrairement à l'approche bayésienne, il n'est pas nécessairement clair quelles connaissances doivent être modélisées. Lors de la modélisation avec des réseaux bayésiens, l'approche est assez claire : selon les dépendances causales, des causes aux effets, un bord après l'autre est entré dans le réseau en testant l'indépendance conditionnelle17. À la fin, tous les CPT sont remplis de valeurs.

Cependant, les combinaisons intéressantes suivantes des deux méthodes sont possibles : nous commençons par construire un réseau selon la méthodologie bayésienne, entrons toutes les arêtes en conséquence, puis remplissons les CPT avec des valeurs. Si certaines valeurs des CPT ne sont pas disponibles, elles peuvent être remplacées par des intervalles ou par d'autres formules logiques probabilistes. Naturellement, un tel réseau - ou mieux : un ensemble de règles - non

17Ce n'est d'ailleurs pas toujours aussi simple.

n'a plus la sémantique particulière d'un réseau bayésien. Il doit ensuite être traité et complété par un système MaxEnt.

La possibilité d'utiliser MaxEnt avec des ensembles de règles arbitraires présente cependant un inconvénient. De manière similaire à la situation en logique, de tels ensembles de règles peuvent être incohérents. Par exemple, les deux règles P (A) = 0,7 et P (A) = 0,8 sont incohérentes. Alors que le système MaxEnt PIT, par exemple, peut reconnaître l'incohérence, s'il ne peut pas donner d'indication sur la façon de résoudre le problème.

Nous avons présenté le système expert médical LEXMED, une application classique pour raisonner avec des connaissances incertaines, et montré comment il peut être modélisé et implémenté à l'aide des réseaux MaxEnt et Bayésiens, et comment ces outils peuvent remplacer le scoring linéaire bien établi, mais trop faible. systèmes utilisés en médecine.18

Dans l' exemple LEXMED , nous avons montré qu'il est possible de construire un système expert de raisonnement dans l'incertitude capable de découvrir (apprendre) des connaissances à partir des données d'une base de données. Nous donnerons plus d'informations sur les méthodes d'apprentissage des réseaux bayésiens au Chap. 8, après avoir jeté les bases nécessaires à l'apprentissage automatique.

Aujourd'hui, le raisonnement bayésien est un vaste domaine indépendant, que nous ne pouvons décrire ici que brièvement. Nous avons complètement laissé de côté la manipulation des variables continues. Pour le cas des variables aléatoires normalement distribuées, il existe des procédures et des systèmes. Pour les distributions arbitraires, cependant, la complexité de calcul est un gros problème.
En plus des réseaux dirigés qui sont fortement basés sur la causalité, il existe également des réseaux non dirigés. En lien avec cela, il y a une discussion sur la signification et l'utilité de la causalité dans les réseaux bayésiens. Le lecteur intéressé est dirigé vers d'excellents manuels tels que [Pea88, Jen01, Whi96, DHS01], ainsi que les actes de la conférence annuelle de l'Association for Uncertainty in Artificial Intelligence (AUAI) (www.auai.org).

7.6 Exercices

Exercice 7.1 Démontrer la proposition du théorème 7.1 page 117.

Exercice 7.2 L'amateur de jardinage Max veut analyser statistiquement sa récolte annuelle de pois. Pour chaque cosse de pois qu'il cueille, il mesure sa longueur xi en centimètres et son poids yi en grammes. Il divise les pois en deux classes, les bons et les mauvais (cosses vides). Les données mesurées (xi, yi) sont

bons petits pois :	x 122334456 et 234455666	mauvais pois :	x 4667 et 2233

18Dans la sect. 8.6 et dans l'exercice 8.16 à la page 219, nous montrerons que les scores sont équivalents au bayes naïf de cas particulier, c'est-à-dire à l'hypothèse que tous les symptômes sont conditionnellement indépendants compte tenu du diagnostic.

(a) À partir des données, calculez les probabilités P (y > 3 |Class = good) et
P (y ≤ 3 | Classe = bon). Utilisez ensuite la formule de Bayes pour déterminer P (Classe =
bien | y > 3) et P (Classe = bon | y ≤ 3).
(b) Laquelle des probabilités calculées dans le sous-problème (a) contredit l'énoncé
"Tous les bons pois pèsent plus de 3 grammes" ?

Exercice 7.3 Vous êtes censé prédire le temps qu'il fera l'après-midi à l'aide de quelques valeurs météorologiques du matin de ce jour. Le calcul de probabilité classique pour cela nécessite un modèle complet, qui est donné dans le tableau suivant.

Ciel	Bar	Préc	P (Ciel, Barre, Préc)
Clair	Montée	Sec	0,40
Clair	Montée	Pluie	0,07
Clair	Chute	sèche	0,08
Clair	Tomber	Pluie	0.10
Bewölkt	Rising	Dry	0,09
Bewölkt	Rising	Raining	0.11
Bewölkt	Chute	sèche	0.03

Ciel : Le ciel est dégagé ou nuageux le matin
Bar : Baromètre en hausse ou tomber le matin
Prec : Pluie ou sec dans le après-midi

(a) Combien y a-t-il d'événements dans la distribution pour ces trois variables ?
(b) Calculer P (Prec = sec|Ciel = clair,Bar = montant).
(c) Calculer P (Prec = pluie|Ciel = nuageux).
(d) Que feriez-vous s'il manquait la dernière ligne du tableau ?

Exercice 7.4 Dans un quiz télévisé, le candidat doit choisir entre trois portes fermées. Derrière une porte, le prix attend : une voiture. Derrière l'un et l'autre les portes sont des chèvres. Le concurrent choisit une porte, par exemple numéro un. L'hôte, qui sait où se trouve la voiture, ouvre une autre porte, par exemple la numéro trois, et une chèvre apparaît. Le concurrent a maintenant la possibilité de choisir entre les deux autres portes (une et deux). Quel est le meilleur choix de son point de vue ? Rester avec la porte qu'il avait initialement choisie ou de passer à l'autre porte fermée ?

Exercice 7.5 En utilisant la méthode des multiplicateurs de Lagrange, montrez que, sans contraintes, la distribution uniforme p1 = p2 =···= pn = 1/n représente l'entropie maximale. N'oubliez pas la contrainte implicitement omniprésente p1 + p2 +···+
pn = 1. Comment pouvons-nous montrer ce même résultat en utilisant l'indifférence ?

Exercice 7.6 Utiliser le système PIT (www.pit-systems.de) pour calculer le MaxEnt solution pour P (B) sous la contrainte P (A) = α et P (B|A) = β. Quel inconvénient du PIT, par rapport au calcul manuel, remarquez-vous ici ?

Exercice 7.7 Étant donné les contraintes P (A) = α et P (A B) = β, calculez manuellement P (B) en utilisant la méthode MaxEnt. Utilisez PIT pour vérifier votre solution.

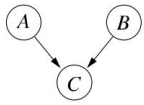
Exercice 7.8 Soit les contraintes de (7.10), (7.11), (7.12) : p1 + p2 = α, p1 + p3 = γ , p1 + p2 + p3 + p4 = 1. Montrer que p1 = αγ, p2 = α (1 − γ), p3 = γ (1 − α), p4 = (1 − α)(1 − γ) représente le maximum d'entropie sous ces contraintes.

Exercice 7.9 Un algorithme probabiliste calcule la probabilité p qu'un email entrant soit un spam. Pour classer les emails dans les classes supprimer et lire, une matrice de coût est ensuite appliquée au résultat. (a) Donnez une matrice de coût (matrice 2 × 2) pour le filtre anti-spam. Supposons ici qu'il en coûte 10 cents à l'utilisateur pour supprimer un e-mail, tandis que la perte d'un e-mail coûte 10 dollars (comparez cela à l'exemple 1.1 à la page 11 et à l'exercice 1.7 à la page 14). (b) Montrer que, pour le cas d'une matrice 2 × 2, l'application de la matrice de coût est équivalente à l'application d'un seuil sur la probabilité de spam et déterminer le seuil.

Exercice 7.10 Soit un réseau bayésien avec les trois variables binaires A,B,C et P (A) = 0,2, P (B) = 0,9, ainsi que le CPT ci-dessous :

(a) Calculer P (A|B). (b) Calculer P (C|A).

PBA (C) wf 0,1
ww 0,2
ff 0,9 ff 0,4



Exercice 7.11 Pour l'exemple d'alarme (Exemple 7.7 à la page 145), calculez les probabilités conditionnelles suivantes : (a) Calculez les probabilités a priori P (Al), P (J), P (M). (b) Calculez P (M|Bur) en utilisant la règle du produit, la marginalisation, la règle de la chaîne et indépendance conditionnelle. (c) Utilisez la formule de Bayes pour calculer P (Bur|M). (d) Calculer P (Al|J,M) et P (Bur|J,M). (e) Montrer que les variables J et M ne sont pas indépendantes. (f) Vérifiez tous vos résultats avec JavaBayes et avec PIT (voir [Ert11] pour la démo programmes).

(g) Concevez un réseau bayésien pour l'exemple d'alarme, mais avec l'ordre des variables modifiées M,Al,Ear,Bur, J . Selon la sémantique des réseaux bayésiens, seules les arêtes nécessaires doivent être dessinées. (Astuce : l'ordre des variables donné ici ne représente PAS la causalité. Il sera donc difficile de déterminer intuitivement les indépendances conditionnelles.)

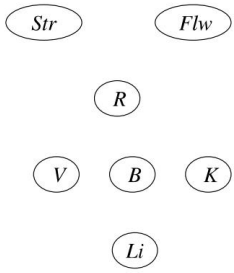
(h) Dans le réseau bayésien d'origine de l'exemple d'alarme, les nœuds sismiques sont supprimés. Quels CPT cela change-t-il ? (Pourquoi ceux-ci en particulier ?)

(i) Calculez le CPT du nœud d'alarme dans le nouveau réseau.

Exercice 7.12 Un système de diagnostic doit être réalisé pour un éclairage de bicyclette alimenté par dynamo en utilisant un réseau bayésien. Les variables du tableau suivant sont données.

Abr.	Ce qui signifie	Valeurs
que Li Light est en		v/f
Str	condition de rue	sec, humide, enneigé
Volant moteur Flw Dynamo usé t/f		
R	Dynamo coulissant t/f	
V	La dynamo affiche la tension t/f	
B	Ampoule ok t/f	
K	Câble ok t/f	

Les variables suivantes sont indépendantes deux à deux : Str, Flw, B, K. De plus : (R, B), (R, K), (V , B), (V , K) sont indépendants et l'équation suivante détient :



$$P(Li|V,R) = P(Li|V)$$

$$P(V|R, Str) = P(V|R)$$

$$P(V|R,Flw) = P(V|R)$$

VBK	P(Li)
ttt	0,99
ttf	0.01
t pi	0,01
tff	0,001
pi t	0,3
pi f	0,005
fft	0,005
fff	0

(a) Dessinez toutes les arêtes dans le graphe (en tenant compte de la causalité).

(b) Entrez tous les CPT manquants dans le graphique (tableau des probabilités conditionnelles). Insérez librement des valeurs plausibles pour les probabilités.

(c) Montrer que le réseau ne contient pas de bord (Str,Li).

(d) Calculez P (V |Str = couvert de neige).

Apprentissage automatique et exploration de données

8

Si nous définissons l'IA comme cela se fait dans le livre d'Elaine Rich [Ric83] :

> L'Intelligence Artificielle est l'étude de la façon de faire faire des choses aux ordinateurs
> moment, les gens vont mieux.

et si l'on considère que la capacité d'apprentissage de l'ordinateur est surtout inférieure à celle des humains, il s'ensuit que la recherche sur les mécanismes d'apprentissage et le développement d'algorithmes d'apprentissage automatique est l'une des branches les plus importantes de l'IA.

Il existe également une demande pour l'apprentissage automatique du point de vue du logiciel développeur qui programme, par exemple, le comportement d'un robot autonome. Le structure du comportement intelligent peut devenir si complexe qu'il est très difficile ou même impossible à programmer de façon presque optimale, même avec des langages modernes de haut niveau tels que PROLOG et Python.1 Des algorithmes d'apprentissage automatique sont même utilisés aujourd'hui pour programmer des robots d'une manière similaire à la façon dont les humains apprennent (voir Chap. 10 ou [BCDS08, RGH+06]), souvent dans un mélange hybride d'être programmé et appris comportement.

La tâche de ce chapitre est de décrire les algorithmes d'apprentissage automatique les plus importants et leurs applications. Le sujet sera introduit dans cette section, suivi par d'importants algorithmes d'apprentissage fondamentaux dans les sections suivantes. La théorie et la terminologie seront construites en parallèle. Le chapitre se terminera par un résumé et aperçu des différents algorithmes et de leurs applications. Nous nous limiterons dans ce chapitre aux algorithmes d'apprentissage supervisés et non supervisés. En tant que classe importante d'algorithmes d'apprentissage, les réseaux de neurones seront traités au Chap. 9.
En raison de sa place particulière et de son rôle important pour les robots autonomes, le renforcement l'apprentissage aura également son propre chapitre dédié (Chap. 10).

Qu'est-ce qu'apprendre ? Apprentissage du vocabulaire d'une langue étrangère, ou technique termes, ou même mémoriser un poème peut être difficile pour beaucoup de gens. Pour les ordinateurs,

1Python est un langage de script moderne avec une syntaxe très lisible, des types de données puissants et des bibliothèques standard étendues, qui peuvent être utilisées à cette fin.

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique,
DOI 10.1007/978-0-85729-299-5_8,

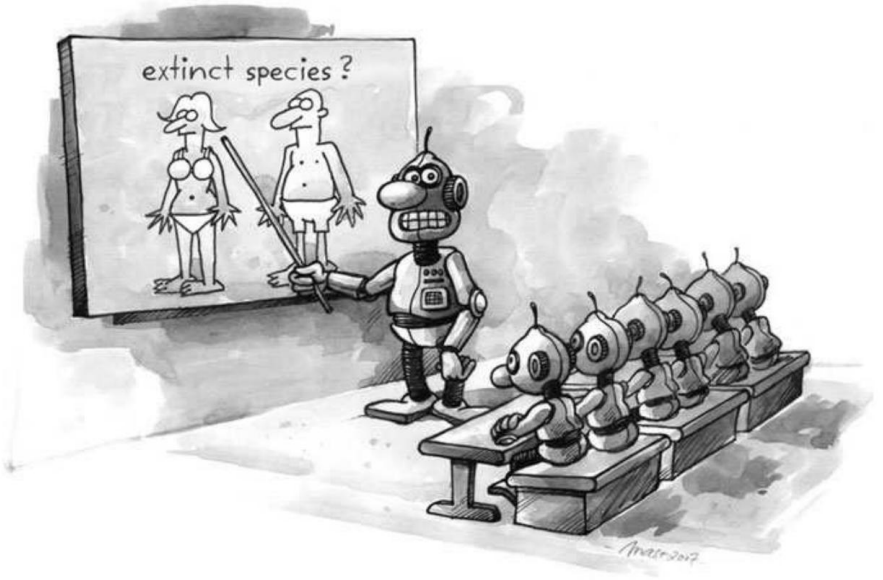


Fig. 8.1 Apprentissage supervisé . . .

cependant, ces tâches sont assez simples car elles ne consistent guère plus qu'à enregistrer du texte dans un fichier. Ainsi la mémorisation est sans intérêt pour l'IA. En revanche, l'acquisition des compétences mathématiques ne se fait généralement pas par mémorisation. Pour l'addition de nombres naturels, cela n'est pas du tout possible, car pour chacun des termes de la somme x + y, il existe une infinité de valeurs. Pour chaque combinaison des deux valeurs x et y, il faudrait stocker le triplet (x, y, x + y), ce qui est impossible. Pour les nombres décimaux, c'est carrément impossible. Cela pose la question : comment apprenons-nous les mathématiques ? La réponse se lit comme suit : L'enseignant explique le processus et les élèves le pratiquent sur des exemples jusqu'à ce qu'ils ne fassent plus d'erreurs sur de nouveaux exemples. Après 50 exemples, l'étudiant (espérons-le) comprend l'addition. C'est-à-dire qu'après seulement 50 exemples, il peut appliquer ce qu'il a appris à une infinité de nouveaux exemples, qui n'avaient pas encore été vus. Ce processus est connu sous le nom de généralisation. Nous commençons par un exemple simple.

Exemple 8.1 Un fruiticulteur souhaite répartir automatiquement les pommes récoltées dans les classes de marchandises A et B. Le dispositif de tri est équipé de capteurs pour mesurer deux caractéristiques, la taille et la couleur, puis décider à laquelle des deux classes appartient la pomme. Il s'agit d'une tâche de classification typique . Les systèmes capables de diviser des vecteurs de caractéristiques en un nombre fini de classes sont appelés classificateurs.

Pour configurer la machine, les pommes sont cueillies à la main par un spécialiste, c'est-à-dire qu'elles sont classées. Ensuite, les deux mesures sont saisies avec leur étiquette de classe dans un tableau (Tableau 8.1 à la page 163). La taille est donnée sous forme de diamètre en centimètres et la couleur par une valeur numérique comprise entre 0 (pour le vert) et 1 (pour le rouge).

Raisonner avec incertitude 7

Nous avons déjà montré dans la Sect. 4 avec le problème de Tweety que la logique à deux valeurs conduit à des problèmes de raisonnement quotidien. Dans cet exemple, les déclarations Titi est un pingouin, Les pingouins sont des oiseaux et Tous les oiseaux peuvent voler conduisent à l'inférence (sémantiquement incorrecte) Titi peut voler. La théorie des probabilités fournit un langage dans lequel nous pouvons formaliser l'énoncé. Presque tous les oiseaux peuvent voler et effectuer des inférences à son sujet. La théorie des probabilités est une méthode éprouvée que nous pouvons utiliser ici car l'incertitude quant à savoir si les oiseaux peuvent voler peut être bien modélisée par une valeur de probabilité. Nous montrerons que des déclarations telles que 99 % de tous les oiseaux peuvent voler, associées à une logique probabiliste, conduisent à des inférences correctes.

Raisonner dans l'incertitude avec des ressources limitées joue un rôle important dans les situations quotidiennes ainsi que dans de nombreuses applications techniques de l'IA. Dans ces domaines, les processus heuristiques sont très importants, comme nous l'avons déjà discuté au Chap. 6. Par exemple, nous utilisons des techniques heuristiques lors de la recherche d'une place de parking dans le trafic urbain. Les heuristiques seules ne suffisent souvent pas, en particulier lorsqu'une décision rapide est nécessaire compte tenu de connaissances incomplètes, comme le montre l'exemple suivant. Un piéton traverse la rue et une auto s'approche rapidement. Pour éviter un accident grave, le piéton doit réagir rapidement. Il n'est pas capable de se soucier d'informations complètes sur l'état du monde, dont il aurait besoin pour les algorithmes de recherche discutés au Chap. 6. Il doit donc parvenir à une décision optimale sous les contraintes données (peu de temps et peu de connaissances potentiellement incertaines). S'il réfléchit trop longtemps, ce sera dangereux. Dans cette situation et dans de nombreuses situations similaires (voir Fig. 7.1 à la page 114), une méthode de raisonnement avec des connaissances incertaines et incomplètes est nécessaire.

Nous voulons étudier les différentes possibilités de raisonnement sous incertitude dans un exemple simple de diagnostic médical. Si un patient ressent une douleur dans le bas-ventre droit et une augmentation du nombre de globules blancs (leucocytes), cela soulève la suspicion qu'il pourrait s'agir d'une appendicite. Nous modélisons cette relation en utilisant la logique propositionnelle avec la formule

Douleur à l'estomac en bas à droite Leucocytes > 10000 → Appendicite

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique,
DOI 10.1007/978-0-85729-299-5_7,

Fig. 7.1 « Asseyons-nous et réfléchissons à ce qu'il faut faire !

Si nous savons alors que

Douleur à l'estomac en bas à droite Leucocytes > 10000

est vrai, alors nous pouvons utiliser le modus ponens pour dériver Appendicite. Ce modèle est clairement trop grossier. En 1976, Shortliffe et Buchanan l'ont reconnu lors de la construction de leur système expert médical MYCIN [Sho76]. Ils ont développé un calcul utilisant des facteurs dits de certitude, qui permettaient de représenter la certitude des faits et des règles.

Une règle A → B se voit attribuer un facteur de certitude β. La sémantique d'une règle A →β B est définie par la probabilité conditionnelle P (B|A) = β. Dans l'exemple ci-dessus, la règle pourrait alors lire

Douleur à l'estomac en bas à droite Leucocytes > 10000 →0,6 Appendicite.

Pour raisonner avec ce genre de formules, ils ont utilisé un calcul pour relier les facteurs de règles. Il s'est avéré, cependant, qu'avec ce calcul, des résultats incohérents pouvaient être obtenus.

Comme discuté au Chap. 4, il y a également eu des tentatives pour résoudre ce problème en utilisant une logique non monotone et une logique par défaut, qui, cependant, ont finalement échoué. La théorie de Dempster-Schäfer attribue une fonction de croyance Bel(A) à une proposition logique A, dont la valeur donne le degré de preuve de la vérité de A. Mais même ce formalisme a des faiblesses, ce qui est montré dans [Pea88] en utilisant une variante de l'exemple de Tweety. Même la logique floue, qui réussit surtout en théorie du contrôle, montre des faiblesses considérables pour raisonner sous incertitude dans des applications plus complexes [Elk93].

Depuis le milieu des années 1980 environ, la théorie des probabilités a de plus en plus d'influence sur l'IA [Pea88, Che85, Whi96, Jen01]. Dans le domaine du raisonnement avec les réseaux bayésiens, ou probabilité subjective, il s'est assuré une place de choix parmi les techniques d'IA à succès. Plutôt que l'implication comme on l'appelle en logique (implication matérielle), on utilise ici la probabilité conditionnelle, qui modélise nettement mieux le raisonnement causal quotidien. Raisonner avec les probabilités profite largement du fait que la théorie des probabilités est une branche des mathématiques bien établie et vieille de plusieurs centaines d'années.

Dans ce chapitre, nous sélectionnerons un point d'entrée élégant, mais pour un manuel d'instructions quelque peu inhabituel, dans ce domaine. Après une brève introduction aux fondements les plus importants nécessaires ici pour raisonner avec des probabilités, nous commencerons par un exemple simple mais important pour raisonner avec des connaissances incertaines et incomplètes. De manière tout à fait naturelle, presque convaincante, nous serons amenés à la méthode d'entropie maximale (MaxEnt). Puis nous montrerons l'utilité de cette méthode en pratique à l'aide du système expert médical LEXMED. Enfin nous introduirons le raisonnement désormais répandu avec les réseaux bayésiens, et montrerons la relation entre les deux méthodes.

7.1 Calcul avec probabilités

Le lecteur qui est familier avec la théorie des probabilités peut sauter cette section. Pour tous les autres, nous donnerons une montée en puissance rapide et recommanderons quelques manuels appropriés tels que [Ros09, FPP07].

La probabilité est particulièrement bien adaptée pour modéliser le raisonnement dans l'incertitude. L'une des raisons en est que les probabilités sont intuitivement faciles à interpréter, comme le montre l'exemple élémentaire suivant.

Exemple 7.1 Pour un seul lancer de dé de jeu (expérience), la probabilité de l'événement « lancer un six » est égale à 1/6, alors que la probabilité de l'occurrence « lancer un nombre impair » est égale à 1/2.

Définition 7.1 Soit Ω l'ensemble fini des événements d'une expérience. Chaque événement ω Ω représente un résultat possible du jet. Si ces événements wi Ω s'excluent mutuellement, mais couvrent tous les résultats possibles de la tentative, alors ils sont appelés événements élémentaires.

Exemple 7.2 Pour un seul lancer d'un dé de jeu

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

car aucun de ces événements ne peut se produire simultanément. Rouler un nombre pair ({2, 4, 6}) n'est donc pas un événement élémentaire, pas plus qu'un nombre inférieur à cinq ({1, 2, 3, 4}) car {2, 4, 6}∩{1, 2, 3, 4}={2, 4} = .

Étant donné deux événements A et B, A B est aussi un événement. Ω lui-même est noté le certain événement impossible, et l'ensemble vide l' événement impossible.

Dans la suite, nous utiliserons la notation logique propositionnelle pour les opérations ensemblistes. Autrement dit, pour l'ensemble A∩B nous écrivons A B. Il ne s'agit pas seulement d'une transformation syntaxique, c'est plutôt sémantiquement correct car l'intersection de deux ensembles est définie comme

$$x \quad \text{UNE} \cap B \quad x \quad \text{UNE} \quad x \quad B.$$

Comme il s'agit de la sémantique de A B, nous pouvons et allons utiliser cette notation. C'est aussi vrai pour les autres opérations d'ensemble union et complément, et nous le ferons, comme indiqué dans le tableau suivant, utilisez également la notation logique propositionnelle pour eux.

Notation ensembliste	Logique propositionnelle	Description
UNE ∩ B	A B	Intersection/et
A B	UNE B	Syndicat/ou
UN	¬A	Complément/négation
Ω	w	Certain événement/vrai
	F	Événement impossible/faux

Les variables utilisées ici (par exemple A, B, etc.) sont appelées variables aléatoires dans la théorie des probabilités. Nous n'utiliserons ici que des variables aléatoires discrètes à domaines finis. La variable face_number pour un lancer de dés est discrète avec les valeurs 1, 2, 3, 4, 5, 6. La probabilité d'obtenir un cinq ou un six est égale à 1/3. Cela peut être décrit par

$$P(\text{numéro\_visage} \quad \{5, 6\}) = P(\text{numéro\_visage} = 5 \quad \text{numéro\_visage} = 6) = 1/3.$$

Le concept de probabilité est censé nous donner une description aussi objective que possible de notre "croyance" ou "conviction" sur le résultat d'une expérience. Tous les nombres dans l'intervalle [0, 1] devraient être possibles, où 0 est la probabilité de événement impossible et 1 la probabilité de l'événement certain. Nous en venons à cela de la définition suivante.

Définition 7.2 Soit Ω = {ω1, ω2,...,ωn} un fini. Il n'y a pas d'événement élémentaire préféré, ce qui signifie que nous supposons une symétrie liée à la fréquence d'apparition de chaque événement élémentaire. La probabilité P (A) de l'événement A est alors

$$P(A) = \frac{|A|}{|\Omega|} = \frac{\text{Nombre de cas favorables pour A}}{\text{Nombre de cas possibles}}.$$

Il s'ensuit immédiatement que tout événement élémentaire a la probabilité 1/|Ω|. L'exigence selon laquelle les événements élémentaires ont une probabilité égale est appelée hypothèse de Laplace et les probabilités ainsi calculées sont appelées probabilités de Laplace. Cette définition atteint sa limite lorsque le nombre d'événements élémentaires devient infini. Parce que nous ne regardons ici que des espaces d'événements finis, cela ne pose pas de problème. Pour décrire les événements, nous utilisons des variables avec le nombre approprié de valeurs. Par exemple, une variable eye_color peut prendre les valeurs green, blue, brown. eye_color = blue décrit alors un événement car on a affaire à une proposition avec les valeurs de vérité t ou f . Pour les variables binaires (booléennes), la variable elle-même est déjà une proposition. Ici il suffit, par exemple, d'écrire P (JohnCalls) au lieu de P (JohnCalls = t).

Exemple 7.3 Selon cette définition, la probabilité de lancer un nombre pair est

$$P(\text{numéro\_visage} \quad \{2, 4, 6\}) = \frac{|\{2, 4, 6\}|}{|\{1, 2, 3, 4, 5, 6\}|} = \frac{3}{6} = \frac{1}{2}.$$

Les règles importantes suivantes découlent directement de la définition.

Théorème 7.1

1. P (Ω) = 1.
2. P ( ) = 0, ce qui signifie que l'événement impossible a une probabilité de 0.
3. Pour les événements exclusifs par paires A et B, il est vrai que P (A B) = P (A) + P (B).
4. Pour deux événements complémentaires A et ¬A il est vrai que P (A) + P (¬A) = 1.
5. Pour des événements arbitraires A et B, il est vrai que P (A B) = P (A) + P (B) − P (A B).
6. Pour A B il est vrai que P (A) ≤ P (B).
7. Si A1,...,An sont des événements élémentaires, $\sum_{i=1}^{n} P(A_i) = 1$ (normalisation alors condition).

L'expression P (A B) ou de manière équivalente P (A, B) représente la probabilité des événements A B. On s'intéresse souvent aux probabilités de toutes les événements, c'est-à-dire de toutes les combinaisons de toutes les valeurs des variables A et B. variables binaires A et B ce sont P (A, B), P (A, ¬B), P (¬A,B), P (¬A, ¬B).
On appelle le vecteur

$$(P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

composée de ces quatre valeurs une distribution ou distribution de probabilité conjointe des variables A et B. Un raccourci pour cela est P(A, B). La distribution dans le cas de deux variables peuvent être bien visualisées sous la forme d'un tableau (matrice), représenté comme suit :

P(A, B)	B = w	B = f
A = w	P (A, B)	P (A,¬B)
A = f	P(¬A,B)	P (¬A,¬B)

Pour les d variables X1,...,Xd à n valeurs chacune, la distribution prend les valeurs P (X1 = x1,...,Xd = xd) et x1,...,xd , qui prennent chacun n valeurs différentes. La distribution peut donc être représentée comme une matrice à d dimensions avec un total de n^d éléments. En raison de la condition de normalisation du théorème 7.1 à la page 117, cependant, l'un de ces n^d valeurs est redondante et la distribution est caractérisée par $n^d - 1$ valeurs uniques.

7.1.1 Probabilité conditionnelle

Exemple 7.4 Sur la rue Landsdowne à Boston, la vitesse de 100 véhicules est mesurée. Pour chaque mesure, il est également noté si le conducteur est un étudiant. Le

les résultats sont

Événement	Fréquence	Fréquence relative
Véhicule observé	100	1
Le chauffeur est étudiant (S)	30	0,3
Vitesse trop élevée (G)	dix	0,1
Le conducteur est étudiant et excès de vitesse (S G)	5	0,05

Nous posons la question : Les élèves accélèrent-ils plus fréquemment que la moyenne par fils, ou que des non-étudiants ?1

La réponse est donnée par la probabilité

$$P(G|S) = \frac{|\text{Le conducteur est étudiant et excès de vitesse}|}{|\text{Le conducteur est étudiant}|} = \frac{5}{30} = \frac{1}{6} \approx 0{,}17$$

1Les probabilités calculées ne peuvent être utilisées pour les propositions continues que si l'échantillon mesuré (100 véhicules) est représentatif. Sinon, seules les propositions sur les 100 véhicules observés peuvent être fait.

pour excès de vitesse à condition que le conducteur soit un étudiant. Ceci est évidemment différent de la probabilité a priori P (G) = 0,1 pour les excès de vitesse. Pour la probabilité a priori, l'espace événementiel n'est pas limité par des conditions supplémentaires.

Définition 7.3 Pour deux événements A et B, la probabilité P (A|B) pour A sous la condition B (probabilité conditionnelle) est définie par

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Dans l'exemple 7.4 , on voit que dans le cas d'un espace d'événements fini, la probabilité conditionnelle P (A|B) peut être comprise comme la probabilité de A ∩ B quand on ne regarde que l'événement B, c'est-à-dire comme

$$P(A|B) = \frac{|A \cap B|}{|B|}.$$

Cette formule peut être facilement dérivée en utilisant la Définition 7.2 à la page 117

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{|A \cap B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{|A \cap B|}{|B|}.$$

Définition 7.4 Si, pour deux événements A et B,

$$P(A|B) = P(A),$$

alors ces événements sont dits indépendants.

Ainsi A et B sont indépendants si la probabilité de l'événement A n'est pas influencée par l'événement B.

Théorème 7.2 Pour les événements indépendants A et B, il résulte de la définition que

$$P(A \cap B) = P(A) \cdot P(B).$$

Exemple 7.5 Pour un lancer de deux dés, la probabilité d'obtenir deux six est de 1/36 si les deux dés sont indépendants car

$$P(D1 = 6 \quad D2 = 6) = P(D1 = 6) \cdot P(D2 = 6) = 6 \qquad \frac{1}{-} \cdot \frac{1}{6} = \frac{1}{36},$$

où la première équation n'est vraie que lorsque les deux dés sont indépendants. Si, par exemple, par un pouvoir magique, le dé 2 est toujours le même que le dé 1, alors

$$P(D1 = 6 \quad D2 = 6) = 6 \quad \frac{1}{-} \cdot$$

Règle de la chaîne

La résolution de la définition de la probabilité conditionnelle pour P (A B) aboutit à la règle dite du produit

$$P(A \quad B) = P(A|B)P(B),$$

que nous généralisons immédiatement au cas de n variables. Par application répétée de la règle ci-dessus, nous obtenons la règle de la chaîne

$$\begin{aligned} &P(X1,...,Xn) \\ &= P(Xn|X1,...,Xn-1) \cdot P(X1,...,Xn-1) \\ &= P(Xn|X1,...,Xn-1) \cdot P(Xn-1|X1,...,Xn-2) \cdot P(X1,...,Xn-2) \\ &= P(Xn|X1,...,Xn-1) \cdot P(Xn-1|X1,...,Xn-2) \cdot \cdots \cdot P(X2|X1) \cdot P(X1) \\ &= \prod_{je=1}^{n} P(Xi|X1 ...,Xi-1), \end{aligned} \tag{7.1}$$

avec lequel nous pouvons représenter une distribution comme un produit de probabilités conditionnelles. Comme la règle de la chaîne est valable pour toutes les valeurs des variables X1,...,Xn, elle a été formulée pour la distribution utilisant le symbole P.

Marginalisation

Parce que A (A B) (A ¬B) est vrai pour les variables binaires A et B

$$P(A) = P((A \quad B) \quad (A \quad \neg B)) = P(A \quad B) + P(A \quad \neg B).$$

Par sommation sur les deux valeurs de B, la variable B est éliminée. De manière analogue, pour des variables arbitraires X1,...,Xd , une variable, par exemple Xd , peut être éliminée par sommation sur toutes leurs variables et on obtient

$$P(X1 = x1,...,Xd-1 = xd-1) = \sum_{xd} P(X1 = x1,...,Xd-1 = xd-1, Xd = xd).$$

L'application de cette formule s'appelle la marginalisation. Cette sommation peut se poursuivre avec les variables X1,...,Xd−1 jusqu'à ce qu'il ne reste qu'une seule variable. Marginalisation peut aussi s'appliquer à la distribution P(X1,...,Xd). La distribution qui en résulte P(X1,...,Xd−1) est appelée distribution marginale. Elle est comparable à la projection d'un cuboïde rectangulaire sur une surface plane. Ici, l'objet tridimensionnel est dessiné sur le bord ou "marge" du cuboïde, c'est-à-dire sur un ensemble à deux dimensions. À la fois cas, la dimensionnalité est réduite de un.

Exemple 7.6 Nous observons l'ensemble de tous les patients qui viennent chez le médecin avec une Douleur d'estomac. Pour chaque patient, la valeur leucocytaire est mesurée, qui est une métrique pour l'abondance relative des globules blancs dans le sang. On définit la variable Leuko, qui est vrai si et seulement si la valeur leucocytaire est supérieure à 10 000. Ce indique une infection dans le corps. Sinon, nous définissons la variable App, qui indique si le patient a une appendicite, c'est-à-dire un appendice infecté. La distribution P(App,Leuko) de ces deux variables est donnée dans le tableau suivant :

P(App,Leuko)	App	¬App	Total
Leuko	0,23	0,31	0,54
¬Leuko	0,05	0,41	0,46
Total	0,28	0,72	1

Dans la dernière ligne, la somme sur les lignes est donnée, et dans la dernière colonne, la somme des colonnes est donnée. Ces sommes sont obtenues par marginalisation. Par exemple, nous lire

$$P(\text{Leuko}) = P(\text{App},\text{Leuko}) + P(\neg\text{App},\text{Leuko}) = 0{,}54.$$

La distribution donnée P(App,Leuko) pourrait provenir d'une enquête auprès de médecins allemands, Par exemple. A partir de là, nous pouvons alors calculer la probabilité conditionnelle

$$P(\text{Leuko}|\text{App}) = \frac{P(\text{Leuko}, \text{App})}{P(\text{application})} = 0{,}82$$

ce qui nous dit qu'environ 82% de tous les cas d'appendicite conduisent à une valeur élevée de leucocytes. Des valeurs comme celle-ci sont publiées dans la littérature médicale. Cependant, la probabilité conditionnelle P (App|Leuko), qui serait en fait beaucoup plus utile pour diagnostiquer appendicite, n'est pas publié. Pour comprendre cela, nous allons d'abord dériver une simple, mais formule très importante.

Théorème de Bayes

Permuter A et B dans la Définition 7.3 à la page 119 donne

$$P(A|B) = \frac{P(A \quad B)}{P(B)} \quad \text{et } P(B|A) = \frac{P(A \quad B)}{P(A)}.$$

En résolvant les deux équations pour P (A B) et en les mettant égales, nous obtenons le théorème de Bayes

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}, \tag{7.2}$$

que nous appliquons immédiatement au problème de l'appendicite et obtenons

$$P(App|Leuko) = \frac{P(Leuko|App) \cdot P(App)}{P(Leuko)} = \frac{0,82 \cdot 0,28}{0,54} = 0,43. \tag{7.3}$$

Maintenant, pourquoi P (Leuko|App) est- il publié, mais pas P (App|Leuko) ?

En supposant que l'appendicite affecte le corps humain sans distinction de race, P (Leuko | App) est une valeur universelle qui est vraie partout dans le monde. Dans (7.3), on voit que P (App|Leuko) n'est pas universel car cette valeur est influencée par les probabilités a priori P (App) et P (Leuko). Chacun d'entre eux peut varier en fonction des circonstances de la vie.
Par exemple, P (Leuko) dépend du fait qu'il y ait beaucoup ou peu de maladies infectieuses dans une population. Cette valeur pourrait potentiellement varier considérablement entre les tropiques et les régions plus froides. Cependant, le théorème de Bayes simplifie le calcul de P (App|Leuko), qui est pertinent pour le diagnostic, à partir de la valeur universellement valide P (Leuko|App).

Avant d'approfondir cet exemple et de l'étendre à un système expert médical pour l'appendicite, nous devons d'abord introduire l'inférence probabiliste nécessaire
mécanisme.

7.2 Le principe d'entropie maximale

Nous allons maintenant montrer, à l'aide d'un exemple d'inférence, qu'un calcul pour raisonner dans l'incertitude peut être réalisé en utilisant la théorie des probabilités. Cependant, nous verrons bientôt que les chemins probabilistes bien usés arrivent rapidement à leur terme. Plus précisément, lorsque trop peu de connaissances sont disponibles pour résoudre les équations nécessaires, de nouvelles idées sont nécessaires. Le physicien américain ET Jaynes a fait un travail de pionnier dans ce domaine dans les années 1950. Il a affirmé qu'étant donné les connaissances manquantes, on peut maximiser l'entropie de la distribution de probabilité souhaitée, et a appliqué ce principe à de nombreux exemples dans [Jay57, Jay03]. Ce principe a ensuite été développé plus avant [Che83, Nil86, Kan89, KK92] et est maintenant mature et peut être appliqué technologiquement, ce que nous montrerons dans l'exemple du projet LEXMED dans Sect. 7.3.

7.2.1 Une règle d'inférence pour les probabilités

Nous voulons dériver une règle d'inférence pour la connaissance incertaine qui est analogue au modus ponens. A partir de la connaissance d'une proposition A et d'une règle A B, la conclusion B sera atteinte. Formulé succinctement, cela se lit

$$\frac{A, A \rightarrow B}{B}.$$

La généralisation des règles de probabilité donne

$$\frac{P(A) = \alpha,\ P(B|A) = \beta}{P(B) = ?}.$$

Soient les deux règles de probabilité α, β données et la valeur P (B) souhaitée. Par marginalisation on obtient la distribution marginale désirée

$$P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A).$$

Les trois valeurs P (A), P (¬A), P (B|A) du côté droit sont connues, mais la valeur P (B|¬A) est inconnue. Nous ne pouvons pas faire une déclaration exacte sur P (B) avec la théorie classique des probabilités, mais tout au plus pouvons-nous estimer P (B) ≥ P (B|A) · P (A).

On considère maintenant la distribution

$$P(A, B) = (P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

et introduisez en sténographie les quatre inconnues

$$p1 = P(A, B),\ p2 = P(A, \neg B),\ p3 = P(\neg A, B),\ p4 = P(\neg A, \neg B).$$

Ces quatre paramètres déterminent la distribution. Si elles sont toutes connues, alors toutes les probabilités pour les deux variables A et B peuvent être calculées. Pour calculer les quatre inconnues, quatre équations sont nécessaires. Une équation est déjà connue sous la forme de la condition de normalisation

$$p1 + p2 + p3 + p4 = 1.$$

Par conséquent, trois équations supplémentaires sont nécessaires. Dans notre exemple, cependant, seules deux équations sont connues.

A partir des valeurs données P (A) = α et P (B|A) = β nous calculons

$$P(A, B) = P(B|A) \cdot P(A) = \alpha\beta$$

et

$$P(A) = P(A, B) + P(A, \neg B).$$

A partir de là, nous pouvons établir le système d'équations suivant et le résoudre dans la mesure du possible :

$$p1 = \alpha\beta, \tag{7.4}$$

$$p1 + p2 = \alpha, \tag{7.5}$$

$$p1 + p2 + p3 + p4 = 1, \tag{7.6}$$

(7.4) dans (7.5): $$p2 = \alpha - \alpha\beta = \alpha(1 - \beta), \tag{7.7}$$

(7.5) dans (7.6) : $$p3 + p4 = 1 - \alpha. \tag{7.8}$$

Les probabilités p1, p2 pour les interprétations (A, B) et (A, ¬B) sont donc connues, mais pour les valeurs p3, p4 il ne reste plus qu'une équation. Pour arriver à une solution définitive malgré cette connaissance manquante, nous changeons de point de vue. Nous utilisons l'équation donnée comme contrainte pour la solution d'un problème d'optimisation.

On cherche une distribution p (pour les variables p3, p4) qui maximise l'entropie

$$H(p) = \sum_{je=1}^{-n} pi \ln pi = -p3 \ln p3 - p4 \ln p4 \tag{7.9}$$

sous la contrainte p3 +p4 = 1−α (7.8). Pourquoi exactement la fonction d'entropie devrait-elle être maximisée ? Comme il nous manque des informations sur la distribution, il faut en quelque sorte les ajouter. Nous pourrions fixer une valeur ad hoc, par exemple p3 = 0,1. Or il vaut mieux déterminer les valeurs p3 et p4 telles que l'information ajoutée soit minimale. On peut montrer (Sect. 8.4.2 et [SW76]) que l'entropie mesure l'incertitude d'une distribution à un facteur constant près. L'entropie négative est alors une mesure de la quantité d'informations que contient une distribution. La maximisation de l'entropie minimise le contenu informationnel de la distribution. Pour visualiser cela, la fonction d'entropie pour le cas bidimensionnel est représentée graphiquement dans la Fig. 7.2 à la page 125.

Pour déterminer le maximum de l'entropie sous la contrainte p3 + p4 − 1 + α = 0 nous utilisons la méthode des multiplicateurs de Lagrange [Ste07]. La fonction de Lagrange lit

$$L = -p3 \ln p3 - p4 \ln p4 + \lambda(p3 + p4 - 1 + \alpha).$$

En prenant les dérivées partielles par rapport à p3 et p4 on obtient

$$\frac{\partial L}{\partial p3} = -\ln p3 - 1 + \lambda = 0,$$

$$\frac{\partial L}{\partial p4} = -\ln p4 - 1 + \lambda = 0$$

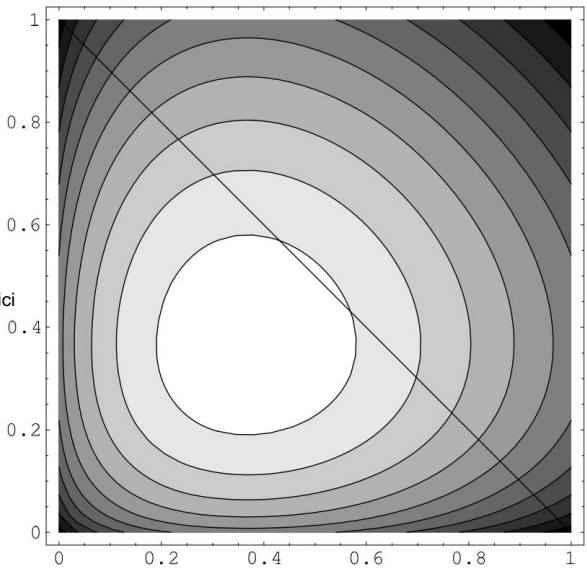


Fig. 7.2 Diagramme de ligne de contour de la fonction d'entropie bidimensionnelle. On voit qu'il est strictement convexe dans tout le carré unité et qu'il a un maximum global isolé. La contrainte p3 + p4 = 1 est également marquée comme cas particulier de la condition p3 + p4 − 1 + α = 0 pour α = 0 qui est pertinente ici

et calculer

$$p3 = p4 = \frac{1 - \alpha}{2}.$$

Nous pouvons maintenant calculer la valeur souhaitée

$$P(B) = P(A, B) + P(\neg A, B) = p1 + p3 = \alpha\beta + = \alpha\ \beta - 2\ \frac{1 - \alpha}{2}\ \frac{1}{2} - + \frac{1}{2}.$$

La substitution en α et β donne

$$P(B) = P(A)\ P(B|A) - 2\ \frac{1}{2}\ 1 + \frac{1}{2}.$$

P (B) est illustré à la Fig. 7.3 à la page 126 pour différentes valeurs de P (B|A). Nous voyons que dans le cas du bord à deux valeurs, c'est-à-dire lorsque P (B) et P (B|A) prennent les valeurs 0 ou 1, l'inférence probabiliste renvoie la même valeur pour P (B) que le modus ponens. Lorsque A et B|A sont tous deux vrais, B est également vrai. Un cas intéressant est P (A) = 0, dans lequel ¬A est vrai. Modus ponens ne peut pas être appliqué ici, mais notre formule donne la valeur 1/2 pour P (B) indépendamment de P (B|A). Lorsque A est faux, nous ne savons rien de B, ce qui reflète exactement notre intuition. Le cas où P (A) = 1 et P (B|A) = 0 est également couvert par la logique propositionnelle. Ici A est vrai et A B faux, et donc A ¬B vrai. Alors B est faux. La ligne horizontale sur la figure signifie que nous ne pouvons pas faire de prédiction sur B dans le cas où P (B|A) = 1/2. Entre ces points, P (B) change linéairement pour les changements de P (A) ou P (B|A).

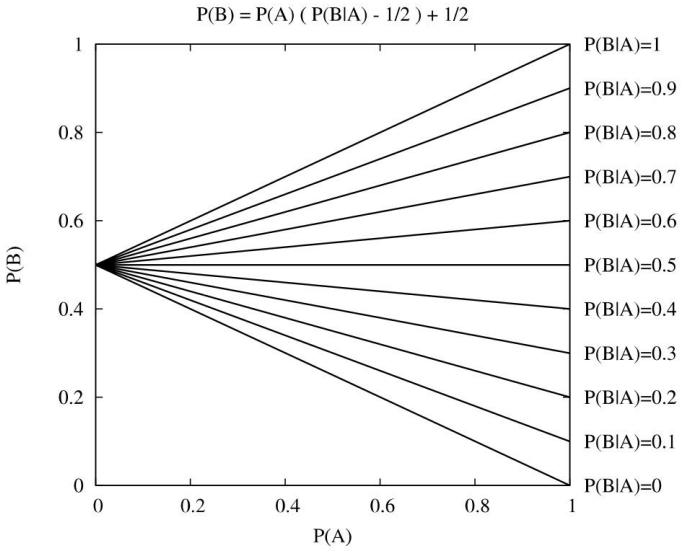


Fig. 7.3 Tableau de courbes pour P (B) en fonction de P (A) pour différentes valeurs de P (B|A)

Théorème 7.3 Soit un ensemble cohérent2 d'équations probabilistes linéaires. Alors il existe un maximum unique pour la fonction d'entropie avec les équations données comme contraintes. La distribution MaxEnt ainsi définie a un contenu minimum d'information sous les contraintes.

Il découle de ce théorème qu'il n'y a pas de distribution qui satisfasse les contraintes et qui ait une entropie plus élevée que la distribution MaxEnt. Un calcul qui conduit à une entropie plus faible ajoute des informations ad hoc supplémentaires, ce qui n'est pas justifié.

En regardant de plus près le calcul ci-dessus de P (B), nous voyons que les deux valeurs p3 et p4 se produisent toujours de manière symétrique. Cela signifie que l'échange des deux variables ne change pas le résultat. Ainsi, le résultat final est p3 = p4. La soi-disant indifférence de ces deux variables conduit à les mettre égales par MaxEnt. Cette relation est valable généralement :

Définition 7.5 Si un échange arbitraire de deux ou plusieurs variables dans les équations de Lagrange aboutit à des équations équivalentes, ces variables sont dites indifférentes.

2Un ensemble d'équations probabilistes est dit cohérent s'il existe au moins une solution, c'est-à-dire une distribution qui satisfait toutes les équations.

Théorème 7.4 Si un ensemble de variables {pi1 ,...,pik } est indifférent, alors le maximum de l'entropie sous les contraintes données est au point où pi1 = pi2 =···= pik .

Avec cette connaissance, nous aurions pu mettre immédiatement les deux variables p3 et p4 égales (sans résoudre les équations de Lagrange).

7.2.2 Entropie maximale sans contraintes explicites

Examinons maintenant le cas où aucune connaissance n'est donnée. Cela signifie que, outre la condition de normalisation

$$p1 + p2 +\cdots+ pn = 1,$$

il n'y a pas de contraintes. Toutes les variables sont donc indifférentes. On peut donc [3] mettez-les égaux et il s'ensuit que p1 = p2 =···= pn = 1/n. Pour raisonner dans l'incertitude, cela signifie qu'en l'absence totale de connaissances, tous les mondes sont également probables. Autrement dit, la distribution est uniforme. Par exemple, dans le cas de deux variables A et B, ce serait le cas que

$$P (A, B) = P (A,\neg B) = P (\neg A,B) = P (\neg A,\neg B) = 1/4,$$

d'où P (A) = P (B) = 1/2 et P (B|A) = 1/2 suivent. Le résultat pour le cas bidimensionnel peut être vu dans la Fig. 7.2 à la page 125 parce que la condition marquée est exactement la condition de normalisation. Nous voyons que le maximum de l'entropie se situe exactement sur la ligne (1/2, 1/2).

Dès que la valeur d'une condition s'écarte de celle dérivée de la distribution uniforme, les probabilités des mondes changent. Nous le montrons dans un autre exemple. Avec les mêmes descriptions que celles utilisées ci-dessus, nous supposons que seuls

$$P (B|A) = \beta$$

est connu. Ainsi P (A, B) = P (B|A)P (A) = βP (A), d'où p1 = β(p1 + p2) découle et on déduit les deux contraintes

$$\beta p2 + (\beta - 1)p1 = 0,$$
$$p1 + p2 + p3 + p4 - 1 = 0.$$

3Le lecteur peut calculer ce résultat en maximisant l'entropie sous la condition de normalisation (Exercice 7.5 page 158).

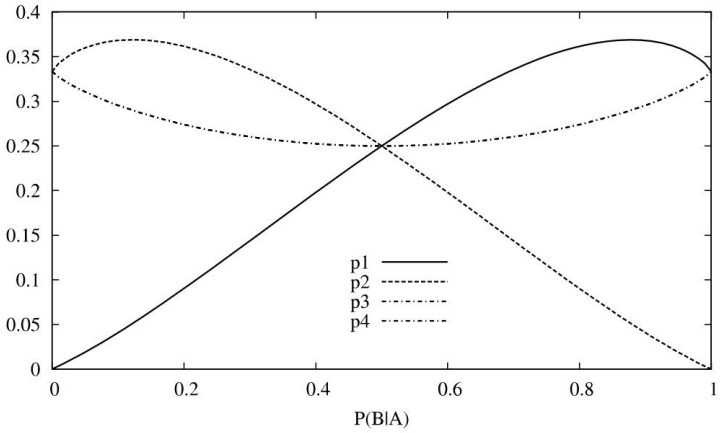


Fig. 7.4 p1, p2, p3, p4 en fonction de β

Tableau 7.1 Table de vérité pour implications matérielles et probabilité conditionnelle pour limite logique propositionnelle

ABA B	P (A)	P (B)	P (B|A)
ttt 111			
t ff 100			
pi w fft 0	0	1	Indéfini
		0	Indéfini

Ici, les équations de Lagrange ne peuvent plus être résolues symboliquement aussi facilement. Une solution numérique des équations de Lagrange donne l'image représentée à la Fig. 7.4, ce qui montre que p3 = p4. Nous pouvons déjà le voir dans les contraintes, dans lesquelles p3 et p4 sont indifférents. Pour P (B|A) = 1/2 on obtient la distribution uniforme, qui n'est pas une surprise. Cela signifie que la contrainte pour cette valeur n'implique pas une restriction sur la distribution. De plus, on voit que pour un petit P (B|A), P (A, B) est aussi petit.

7.2.3 Probabilité conditionnelle versus implication matérielle

Nous allons maintenant montrer que, pour modéliser le raisonnement, la probabilité conditionnelle est préférable que ce qui est connu en logique sous le nom d'implication matérielle (à cette fin, voir aussi [Ada75]). Nous observons d'abord la table de vérité présentée dans le tableau 7.1, dans laquelle la probabilité conditionnelle et l'implication matérielle pour les cas extrêmes de probabilités zéro et un sont par rapport. Dans les deux cas avec de fausses prémisses (qui, intuitivement, sont des cas critiques), P (B|A) est indéfini, ce qui est logique.

Maintenant, nous nous demandons quelle valeur est prise par P (B|A) lorsque des valeurs arbitraires P (A) = α et P (B) = γ sont données et qu'aucune autre information n'est connue. Encore une fois, nous maximisons l'entropie sous les contraintes données. Comme ci-dessus nous posons

$$p1 = P(A, B),\ p2 = P(A, \neg B),\ p3 = P(\neg A, B),\ p4 = P(\neg A, \neg B)$$

et obtenir comme contraintes

$$p1 + p2 = \alpha, \tag{7.10}$$

$$p1 + p3 = \gamma, \tag{7.11}$$

$$p1 + p2 + p3 + p4 = 1. \tag{7.12}$$

Avec cela, nous calculons en utilisant la maximisation de l'entropie (voir l'exercice 7.8 à la page 159)

$$p1 = \alpha\gamma,\ p2 = \alpha(1 - \gamma),\ p3 = \gamma(1 - \alpha),\ p4 = (1 - \alpha)(1 - \gamma).$$

De p1 = αγ il s'ensuit que P (A, B) = P (A) · P (B), ce qui signifie que A et B sont indépendants. Puisqu'il n'y a pas de contraintes reliant A et B, le principe MaxEnt aboutit à l'indépendance de ces variables. La moitié droite du tableau 7.1 à la page 128 facilite la compréhension. De la définition

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

il s'ensuit pour le cas P (A) = 0, c'est-à-dire lorsque la prémisse n'est pas fausse, car A et B sont indépendants, que P (B|A) = P (B). Pour le cas P (A) = 0, P (B|A) reste indéfini.

7.2.4 Systèmes MaxEnt

Comme mentionné précédemment, en raison de la non-linéarité de la fonction d'entropie, l'optimisation MaxEnt ne peut généralement pas être effectuée symboliquement pour des problèmes non triviaux. Ainsi, deux systèmes ont été développés pour la maximisation de l'entropie numérique. Le premier système, SPIRIT (Symmetrical Probabilistic Intensional Reasoning in Inference Net works in Transition, www.xspirit.de), [RM96] a été construit à la Fernuniversität Hagen.
Le second, PIT (Probability Induction Tool) a été développé à l'Université Technique de Munich [Sch96, ES99, SE00]. Nous allons maintenant présenter brièvement le PIT.

Le système PIT utilise la méthode de programmation quadratique séquentielle (SQP) pour trouver un extremum de la fonction d'entropie sous les contraintes données. En entrée, PIT attend des données contenant les contraintes. Par exemple, les contraintes P (A) = α et P (B|A) = β de la Sect. 7.2.1 avoir la forme

```
var A{t,f}, B{t,f} ;

P([A=t]) = 0,6 ;
P([B=t] | [A=t]) = 0,3 ;

QP([B=t]);
QP([B=t] | [A=t]);
```

Comme PIT effectue un calcul numérique, nous devons entrer des valeurs de probabilité explicites. L'avant-dernière ligne contient la requête QP([B=t]). Cela signifie
que P (B) est la valeur souhaitée. Sur www.pit-systems.de sous "Exemples", nous
placez cette entrée dans une page d'entrée vierge ("Blank Page") et démarrez PIT. En conséquence nous
obtenir

Nr.	Valeur de vérité	Requête de probabilité
1	NON SPÉCIFIÉ 3.800e–01	QP([B = t]);
2	NON SPÉCIFIÉ 3.000e–01	QP([A = t]-|> [B = t]);

et à partir de là lire P (B) = 0,38 et P (B|A) = 0,3.

7.2.5 L'exemple de Tweety

Nous montrons maintenant, en utilisant l'exemple Tweety de Sect. 4.3, ce raisonnement probabiliste
et en particulier MaxEnt sont non monotones et modélisent très bien le raisonnement quotidien.
Bien. Nous modélisons les règles pertinentes avec des probabilités comme suit :

P (oiseau|pingouin) = 1	"les pingouins sont des oiseaux"
P (mouches|oiseau) [0,95, 1]	"(presque tous) les oiseaux peuvent voler"
P (mouches|pingouin) = 0	"les pingouins ne savent pas voler"

Les première et troisième règles représentent des prédictions fermes, qui peuvent aussi être facilement
formulées en logique. Dans le second, cependant, nous exprimons notre connaissance que presque tous
les oiseaux peuvent voler au moyen d'un intervalle de probabilité. Avec les données d'entrée PIT

```
var pingouin{oui,non}, oiseau{oui,non}, mouches{oui,non} ;

P([oiseau=oui] | [pingouin=oui]) = 1 ;
P([vole=oui] | [oiseau=oui]) IN [0.95,1] ;
P([mouches=oui] | [pingouin=oui]) = 0 ;

QP([vole=oui]| [pingouin=oui]);
```

nous obtenons la bonne réponse

Nr.	Valeur de vérité	Requête de probabilité
1	NON SPÉCIFIÉ 0.000e+00	QP([pingouin=oui]-|> [mouches=oui]);

avec la proposition que les pingouins ne peuvent pas voler.4 L'explication en est très simple. Avec P (mouches|oiseau) [0.95, 1] il est possible qu'il y ait des oiseaux non volants. Si ce la règle a été remplacée par P (mouches|oiseau) = 1, alors PIT ne pourrait rien faire et afficherait un message d'erreur sur les contraintes incohérentes.

Dans cet exemple, nous pouvons facilement voir que les intervalles de probabilité sont souvent très utiles pour modéliser notre ignorance sur les valeurs de probabilité exactes. On aurait pu faire un formulation encore plus floue de la deuxième règle dans l'esprit de "normalement les oiseaux volent" avec P (vole|oiseau) (0,5, 1]. L'utilisation de l'intervalle semi-ouvert exclut la valeur 0,5.

Il a déjà été montré dans [Pea88] que cet exemple peut être résolu en logique probabiliste, même sans MaxEnt. Dans [Sch96], il est montré pour un certain nombre de benchmarks exigeants pour le raisonnement non monotone que ceux-ci peuvent être résolus élégamment avec MaxEnt. Dans la section suivante, nous présentons une application pratique réussie de MaxEnt sous la forme d'un système expert médical.

7.3 LEXMED, un système expert pour le diagnostic de l'appendicite

Le système expert médical LEXMED, qui utilise la méthode MaxEnt, a été développé à l'Université des sciences appliquées de Ravensburg-Weingarten par Manfred Schramm, Walter Rampf et l'auteur, en collaboration avec le Weingarten 14-Nothelfer Hospital [SE00, Le999].5 L'acronyme LEXMED signifie système expert d'apprentissage pour le diagnostic médical.

7.3.1 Diagnostic d'appendicite avec des méthodes formelles

La cause grave la plus fréquente de douleur abdominale aiguë [dD91] est l'appendicite— une inflammation de l'appendice, un tube à extrémité aveugle relié au caecum. Même aujourd'hui, le diagnostic peut être difficile dans de nombreux cas [OFY+95]. Par exemple, jusqu'à environ 20% des appendices enlevés sont sans signes pathologiques, ce qui signifie que les opérations étaient inutiles. De même, il y a régulièrement des cas où un appendice enflammé n'est pas reconnu comme tel.

Dès le début des années 1970, on a tenté d'automatiser le diagnostic d'appendicite, dans le but de réduire le taux de faux diagnostics [dDLS+72, OPB94, OFY +95]. Le système expert est particulièrement remarquable

4QP([penguin=yes]-|> [flies=yes]) est une forme alternative de la syntaxe PIT pour QP([vole=oui] | [pingouin=oui]).

5Le projet a été financé par le Land allemand de Baden-Württemberg, la compagnie d'assurance maladie AOK Baden-Württemberg, l'Université des Sciences Appliquées de Ravensburg-Weingarten et l'hôpital 14 Nothelfer à Weingarten.

pour le diagnostic des douleurs abdominales aiguës, développé par de Dombal en Grande-Bretagne. Il a été rendu public en 1972, donc nettement antérieur au fameux système MYCIN.

Presque tous les processus de diagnostic formels utilisés en médecine à ce jour ont été basés sur des scores. Les systèmes de cotation sont extrêmement simples à appliquer : Pour chaque valeur d'un symptôme (par exemple fièvre ou mal de ventre en bas à droite) le médecin note un certain nombre de points. Si la somme des points est supérieure à une certaine valeur (seuil), une certaine décision est recommandée (par exemple opération). Pour n symptômes S1,...,Sn, un score d'appendicite peut être décrit formellement comme

$$\text{Diagnostiquer} = \begin{cases} \text{Appendicite si } w1S1 + \cdots + wnSn > \Theta, \text{ négatif} \\ \text{autre.} \end{cases}$$

Avec les scores, une combinaison linéaire de valeurs de symptômes est ainsi comparée à un seuil Θ. Les poids des symptômes sont extraits de bases de données à l'aide de méthodes statistiques. L'avantage des partitions est leur simplicité d'application. La somme pondérée des points peut être facilement calculée à la main et un ordinateur n'est pas nécessaire pour le diagnostic.

En raison de la linéarité de cette méthode, les scores sont trop faibles pour modéliser des relations complexes. La contribution wiSi d'un symptôme Si au score étant calculée indépendamment des autres symptômes, il est clair que les systèmes de score ne peuvent tenir compte d'aucun « contexte ». Principalement, ils ne peuvent pas faire la distinction entre des combinaisons de symptômes, par exemple ils ne peuvent pas faire la distinction entre le nombre de globules blancs d'un patient âgé et celui d'un jeune patient.

Pour un ensemble donné de symptômes, la probabilité conditionnelle est beaucoup plus puissante que les scores pour faire des prédictions car ces derniers ne peuvent pas décrire les dépendances entre différents symptômes. Nous pouvons montrer que les scores supposent implicitement que tous les symptômes sont indépendants.

Lors de l'utilisation des partitions, un autre problème se pose. Pour arriver à une bonne qualité de diagnostic, il faut imposer des exigences strictes aux bases de données utilisées pour déterminer statistiquement les poids wi . En particulier, ils doivent être représentatifs de l'ensemble des patients de la zone dans laquelle le système de diagnostic est utilisé. Ceci est souvent difficile, voire impossible, à garantir. Dans de tels cas, les scores et autres méthodes statistiques ne peuvent pas être utilisés ou auront un taux d'erreurs élevé.

7.3.2 Base de connaissances probabiliste hybride

Des relations probabilistes complexes apparaissent fréquemment en médecine. Avec LEXMED, ces relations peuvent être bien modélisées et calculées rapidement. Ici, l'utilisation de propositions probabilistes, avec lesquelles des informations incertaines et incomplètes peuvent être exprimées et traitées de manière intuitive et mathématiquement fondée, est essentielle. La question suivante peut servir de requête typique contre le système expert : « Quelle est la probabilité d'un appendice enflammé si le patient est un homme de 23 ans souffrant de douleurs dans le bas-ventre droit et d'un nombre de globules blancs de 13 000 ? ?" Formulé sous forme de probabilité conditionnelle, en utilisant les noms et les plages de valeurs des symptômes utilisés dans le tableau 7.2 à la page 133, cela se lit

Tableau 7.2 Symptômes utilisés pour la requête dans LEXMED et leurs valeurs. Le nombre de valeurs pour chaque symptôme est indiqué dans la colonne marquée #

Symptôme	Valeurs	# Court
Genre	Mâle, femelle	2 Sexe2
Âge	0–5, 6–10, 11–15, 16–20, 21–25, 26–35, 36–45, 46–55, 56–65, 65–	10 Âge10
Douleur 1er Quad.	Oui, non	2 P1Q2
Douleur 2ème Quad.	Oui, non	2 P2Q2
Douleur 3ème Quad.	Oui, non	2 P3Q2
Douleur 4ème Quad.	Oui, non	2 P4Q2
Garde	Local, global, aucun	3 Gua3
Rebond tendresse	Oui, non	2 Reb2
Douleur au tapotement	Oui, non	2 Appuyez2
Douleur rectale	Oui, non	2 RecP2
Bruits intestinaux	Faible, normal, augmenté, aucun	4 ArcS4
Échographie anormale	Oui, non	2 Sono2
Sedim urinaire anormal.	Oui, non	2 Urine2
Température (rectale)	–37,3, 37,4–37,6, 37,7–38,0, 38,1–38,4, 38,5–38,9, 39.0–	6 TRec6
Leucocytes	0–6k, 6k–8k, 8k–10k, 10k–12k, 12k–15k, 15k–20k, 20k–	7 Leuko7
Diagnostic	Enflammé, perforé, négatif, autre	4 Diag4

P (Diag4 = enflammé Diag4 = perforé |
Sexe2 = masculin Age10 21–25 Leuko7 12k–15k).

En utilisant des propositions probabilistes, LEXMED a la capacité d'utiliser des informations à partir de bases de données non représentatives car ces informations peuvent être complétées convenablement à partir d'autres sources. LEXMED est sous-jacente à une base de données qui contient des données sur les patients dont les appendices ont été enlevés chirurgicalement. Avec les méthodes statistiques, (environ 400) règles sont générées qui compilent les connaissances contenues dans la base de données sous une forme abstraite [ES99]. Parce qu'il n'y a pas de patients dans cette base de données qui étaient suspectées d'avoir une appendicite mais avaient des diagnostics négatifs (c'est-à-dire ne nécessitant pas de traitement),6 il n'y a aucune connaissance sur les patients dans la base de données. Ainsi, des connaissances provenant d'autres sources doivent être ajoutées. Dans LEXMED , les règles recueillies à partir de la base de données sont donc complétées par (environ 100) règles d'experts médicaux et de la littérature médicale. Cela se traduit par une base de données probabiliste hybride, qui contient des connaissances extraites des données ainsi que des connaissances explicitement formulées par des experts. Parce que les deux types de règles sont

6Ces diagnostics négatifs sont notés « douleurs abdominales non spécifiques » (NSAP).

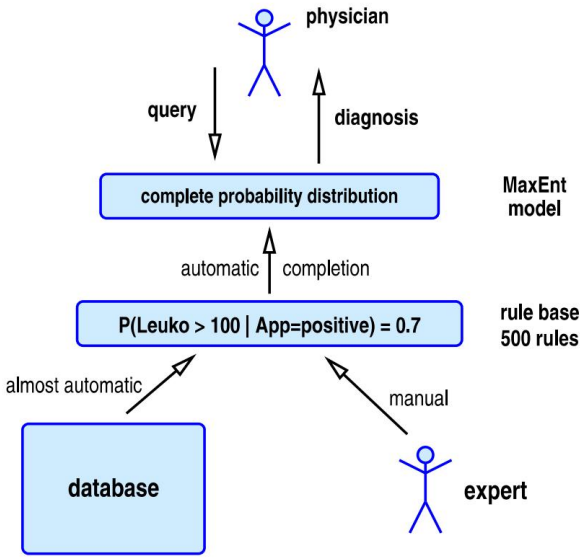


Fig. 7.5 Les règles probabilistes sont générées à partir de données et de connaissances d'experts, qui sont intégrées dans une base de règles (base de connaissances) et finalement complétées à l'aide de la méthode MaxEnt

formulées sous forme de probabilités conditionnelles (voir par exemple (7.14) à la page 138), elles peuvent être facilement combinées, comme le montre la Fig. 7.5 et avec plus de détails à la Fig. 7.7 à la page 136.

LEXMED calcule les probabilités de divers diagnostics en utilisant la distribution de probabilité de toutes les variables pertinentes (voir Tableau 7.2 à la page 133). Étant donné que les 14 symptômes utilisés dans LEXMED et les diagnostics sont modélisés sous forme de variables discrètes (même les variables continues telles que la valeur des leucocytes sont divisées en plages), la taille de la distribution (c'est-à-dire la taille de l'espace des événements) peut être déterminée à l'aide du tableau 7.2 à la page 133 comme le produit du nombre de valeurs de tous les symptômes, ou

$$10\,2 \cdot 10 \cdot 3 \cdot 4 \cdot 6 \cdot 7 \cdot 4 = 20\,643\,840$$

éléments. En raison de la condition de normalisation du théorème 7.1, il a donc 20 643 839 valeurs indépendantes. Chaque ensemble de règles avec moins de 20 643 839 valeurs de probabilité ne décrit potentiellement pas complètement cet espace d'événements. Pour pouvoir répondre à n'importe quelle requête arbitraire, le système expert a besoin d'une distribution complète. La construction d'une distribution aussi étendue et cohérente à l'aide de méthodes statistiques est très difficile7. Exiger d'un expert humain les 20 643 839 valeurs pour la distribution (au lieu des 100 règles susmentionnées) serait essentiellement impossible.

Ici, la méthode MaxEnt entre en jeu. La généralisation d'environ 500 règles à un modèle de probabilité complet se fait dans LEXMED en maximisant l'entropie avec les 500 règles comme contraintes. Un codage efficace de la distribution MaxEnt résultante conduit à des temps de réponse pour le diagnostic d'environ une seconde.

7La tâche consistant à générer une fonction à partir d'un ensemble de données est connue sous le nom d'apprentissage automatique. Nous couvrirons cela en détail au Chap. 8.

Personal Details	unknown	values	
Gender	◉	○ male ○ female	?
Age-group	○	○ 0-5 ○ 6-10 ○ 11-15 ○ 16-20 ◉ 21-25 ○ 26-35 ○ 36-45 ○ 46-55 ○ 56-65 ○ 65-	?
Results of examination	**not done**	**values**	
1st quadrant	◉	○ yes ○ no	?
2nd quadrant	◉	○ yes ○ no	?
3rd quadrant	○	◉ yes ○ no	?
4th quadrant	◉	○ yes ○ no	?
guarding	○	◉ local ○ global ○ none	?
rebound tenderness	○	◉ yes ○ no	?
pain on tapping	○	◉ yes ○ no	?
rectal pain	◉	○ yes ○ no	?
bowel sounds	○	○ weak ◉ normal ○ increased ○ none	?
abnormal ultrasound	◉	○ yes ○ no	?
abnormal urine sediment	◉	○ yes ○ no	?
temperature range (rectal)	○	○ -37.3 ○ 37.4-37.6 ○ 37.7-38.0 ○ 38.1-38.4 ◉ 38.5-38.9 ○ 39.0-	?
leucocyte count	○	○ 0-6k ○ 6k-8k ○ 8k-10k ○ 10k-12k ◉ 12k-15k ○ 15k-20k ○ 20k-	?

	Result of the PIT diagnosis			
Diagnosis	App. inflamed	App. perforated	Negative	Other
Probability	0.70	0.17	0.06	0.07

Fig. 7.6 Le masque de saisie LEXMED pour la saisie des symptômes examinés et en dessous la sortie des probabilités de diagnostic résultantes

7.3.3 Application de LEXMED

L'utilisation de LEXMED est simple et explicite. Le médecin visite le Pour un diagnostic
Page d'accueil de LEXMED sur www.lexmed.de[8].automatique, le médecin
saisit les résultats de son examen dans le formulaire de saisie de la Fig. 7.6. Après une ou deux secondes, il reçoit les probabilités pour les quatre diagnostics différents ainsi qu'une proposition de traitement (Sect. 7.3.5). Si certains résultats d'examen manquent en entrée (par exemple les résultats de l'échographie), alors le médecin choisit l'entrée non examinée. Naturellement, la certitude du diagnostic est plus élevée lorsque davantage de valeurs de symptômes sont saisies.

La version 8A aux fonctionnalités limitées est accessible sans mot de passe.

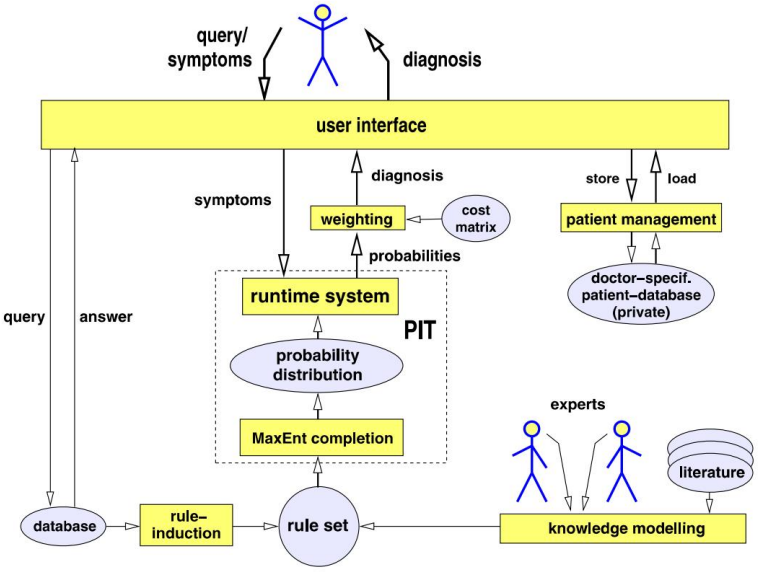


Fig. 7.7 Les règles sont générées à partir de la base de données ainsi que des connaissances d'experts. À partir de ceux-ci, MaxEnt crée une distribution de probabilité complète. Pour une requête utilisateur, la probabilité de chaque diagnostic possible est calculée. À l'aide de la matrice des coûts (voir section 7.3.5), une décision est alors suggérée

Chaque utilisateur enregistré a accès à une base de données de patients privée, dans laquelle les données d'entrée peuvent être archivées. Ainsi, les données et les diagnostics des patients antérieurs peuvent être facilement comparés à ceux d'un nouveau patient.

7.3.4 Fonction de LEXMED

La connaissance est formalisée à l'aide de propositions probabilistes. Par exemple, la proposition

$$P\,(\text{Leuko7} > 20000 \mid \text{Diag4} = \text{enflammé}) = 0{,}09$$

donne une fréquence de 9 % pour une valeur leucocytaire supérieure à 20 000 en cas d'appendice inflammatoire.9

Apprentissage des règles par induction statistique Les données brutes de la base de données de LEXMED contiennent 54 valeurs différentes (anonymisées) pour 14 646 patients. Comme mentionné précédemment, seuls les patients dont les appendices ont été enlevés chirurgicalement sont inclus dans cette base de données. Sur les 54 attributs utilisés dans la base de données, après une analyse statistique, les 14 symptômes présentés dans le tableau 7.2 à la page 133

[9] Au lieu de valeurs numériques individuelles, des intervalles peuvent également être utilisés ici (par exemple [0,06, 0,12]).

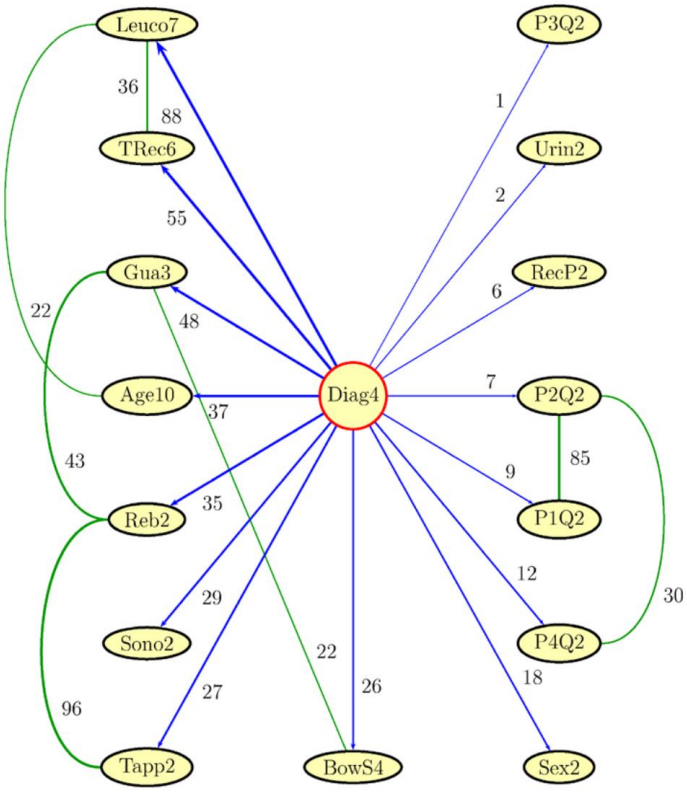


Fig. 7.8 Graphique de dépendance calculé à partir de la base de données

ont été utilisées. Maintenant, les règles sont créées à partir de cette base de données en deux étapes. La première étape détermine la structure de dépendance des symptômes. La deuxième étape remplit cette structure avec les règles de probabilité respectives.10

Détermination du graphe de dépendance Le graphe de la Fig. 7.8 contient pour chaque variable (le symptôme et le diagnostic) un nœud et des arêtes dirigées qui relient différents nœuds. L'épaisseur des bords entre les variables représente une mesure de la dépendance statistique ou de la corrélation des variables. La corrélation de deux variables indépendantes est égale à zéro. La corrélation de paires pour chacun des 14 symptômes avec Diag4 a été calculée et répertoriée dans le graphique. De plus, toutes les triples corrélations entre le diagnostic et deux symptômes ont été calculées. Parmi ceux-ci, seules les valeurs les plus fortes ont été tracées comme des bords supplémentaires entre les deux symptômes participants.

10Pour une introduction systématique au machine learning nous renvoyons le lecteur au Chap. 8.

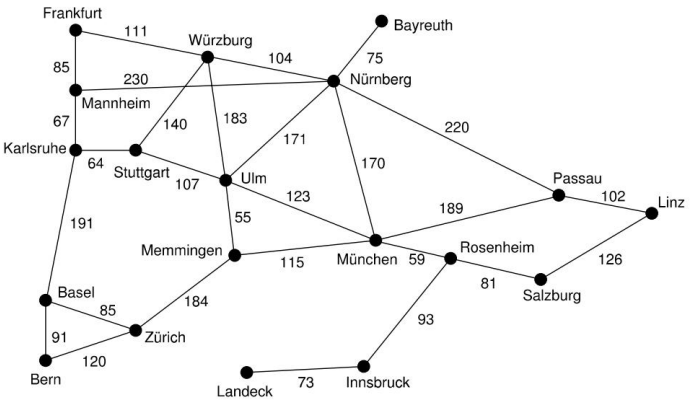


Fig. 6.5 Le graphique du sud de l'Allemagne comme exemple d'une tâche de recherche avec une fonction de coût

pour b car un arbre avec un facteur de branchement constant et une profondeur d a un total de

$$n = \sum_{je=0}^{d} je\,b = \frac{b\,d+1 - 1}{b-1} \tag{6.2}$$

nœuds.

Pour l'application pratique des algorithmes de recherche pour les arbres de recherche finis, le dernier niveau est particulièrement important car

> Théorème 6.1 Pour les arbres de recherche finis fortement ramifiés avec un grand facteur de branchement constant, presque tous les nœuds sont au dernier niveau.

La démonstration simple de ce théorème est recommandée au lecteur comme exercice (Exercice 6.1 à la page 110).

Exemple 6.2 On nous donne une carte, telle que celle représentée sur la Fig. 6.5, sous forme de graphe avec les villes comme nœuds et les connexions routières entre les villes comme arêtes pondérées avec des distances. Nous recherchons un itinéraire optimal de la ville A à la ville B. La description du schéma correspondant indique État : une ville comme emplacement
actuel du voyageur.
Etat de départ : Une ville arbitraire.
État du but : Une ville arbitraire.
Actions : Voyagez de la ville actuelle vers une ville voisine.
Fonction de coût : La distance entre les villes. Chaque action correspond à une arête
dans le graphique avec la distance comme poids.
Espace d'état : toutes les villes, c'est-à-dire les nœuds du graphe.
Pour trouver l'itinéraire avec une longueur minimale, les coûts doivent être pris en compte car ils ne sont pas constants comme ils l'étaient dans le 8-puzzle.

Définition 6.3 Un algorithme de recherche est dit optimal s'il, si une solution existe, trouve toujours la solution de moindre coût.

Le problème des 8 énigmes est déterministe, ce qui signifie que chaque action mène d'un état à un état successeur unique. Il est en outre observable, c'est-à-dire que l'agent sait toujours dans quel état il se trouve. Dans la planification d'itinéraire dans des applications réelles, les deux caractéristiques ne sont pas toujours données. L'action « Conduire de Munich à Ulm » peut, par exemple à cause d'un accident, conduire à l'état successeur « Munich ». Il peut aussi arriver que le voyageur ne sache plus où il se trouve parce qu'il s'est perdu. Nous voulons ignorer ce genre de complications dans un premier temps. Par conséquent, dans ce chapitre, nous ne traiterons que des problèmes déterministes et observables.

Des problèmes comme le 8-puzzle, qui sont déterministes et observables, rendent la planification d'action relativement simple car, grâce à un modèle abstrait, il est possible de trouver des séquences d'action pour la solution du problème sans réellement effectuer les actions dans le monde réel. . Dans le cas du 8-puzzle, il n'est pas nécessaire de déplacer réellement les cases dans le monde réel pour trouver la solution. Nous pouvons trouver des solutions optimales avec des algorithmes dits hors ligne. On est confronté à des défis très différents lorsque, par exemple, la construction de robots censés jouer au football. Ici, il n'y aura jamais de modèle abstrait exact des actions. Par exemple, un robot qui frappe le ballon dans une direction spécifique ne peut pas prédire avec certitude où le ballon se déplacera car, entre autres, il ne sait pas si un adversaire va attraper ou dévier le ballon. Ici, des algorithmes en ligne sont alors nécessaires, qui prennent des décisions basées sur les signaux des capteurs dans chaque situation. Apprentissage par renforcement, décrit dans la Sect. 10, travaille à l'optimisation de ces décisions en fonction de l'expérience.

6.2 Recherche non informée

6.2.1 Recherche étendue d'abord

Dans la recherche en largeur d'abord, l'arbre de recherche est exploré de haut en bas selon l'algorithme donné à la Fig. 6.6 à la page 90 jusqu'à ce qu'une solution soit trouvée. Tout d'abord, chaque nœud de la liste de nœuds est testé pour savoir s'il s'agit d'un nœud cible, et en cas de succès, le programme est arrêté. Sinon, tous les successeurs du nœud sont générés. La recherche se poursuit alors récursivement sur la liste de tous les nœuds nouvellement générés. Le tout se répète jusqu'à ce qu'il n'y ait plus de successeurs générés.

Cet algorithme est générique. Autrement dit, cela fonctionne pour des applications arbitraires si les deux fonctions spécifiques à l'application "GoalReached" et "Successors" sont fournies. "GoalReached" calcule si l'argument est un nœud d'objectif, et "Successors" calcule la liste de tous les nœuds successeurs de son argument. La Figure 6.7 à la page 90 montre un instantané de la recherche en largeur d'abord.

```
BREADTH-FIRST-SEARCH (NodeList, Goal)
NouveauNoeuds =
Pour tout Node      NodeList
    Si objectif atteint (nœud, objectif)
        Retour("Solution trouvée", Nœud)
    NouveauxNoeuds = Ajouter(NouveauxNoeuds, Successeurs(Noeud))
Si NouveauNoeuds =
    Retour(BREADTH-FIRST-SEARCH(NewNodes, Goal))
Autre
    Retour ("Pas de solution")
```

Fig. 6.6 L'algorithme pour la recherche en largeur d'abord

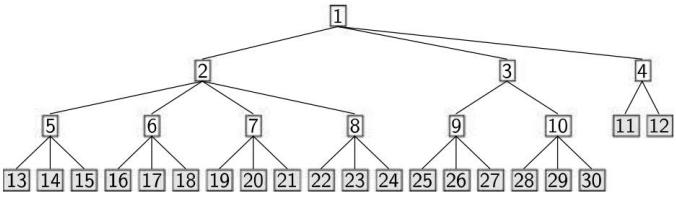


Fig. 6.7 Recherche en largeur d'abord lors de l'expansion des nœuds de troisième niveau. Les nœuds sont numérotés selon l'ordre dans lequel ils ont été générés. Les successeurs des nœuds 11 et 12 n'ont pas encore été générés

Analyse Étant donné que la recherche en largeur d'abord recherche complètement à travers chaque profondeur et atteint chaque profondeur en un temps fini, elle est complète si le facteur de branchement b est fini. La solution optimale (c'est-à-dire la plus courte) est trouvée si les coûts de toutes les actions sont les mêmes (voir l'exercice 6.7 à la page 110). Le temps de calcul et l'espace mémoire croissent de manière exponentielle avec la profondeur de l'arbre. Pour un arbre avec un facteur de branchement b et une profondeur d constants, le temps de calcul total est donc donné par

$$c \cdot \sum_{je=0}^{d} je\, b = \frac{b\, d+1 - 1}{b-1} = O(bd\ ).$$

Bien que seul le dernier niveau soit enregistré en mémoire, l'espace mémoire requis est également O(bd).

Avec la vitesse des ordinateurs d'aujourd'hui, qui peuvent générer des milliards de nœuds en quelques minutes, la mémoire principale se remplit rapidement et la recherche se termine. Le problème de la solution la plus courte qui n'est pas toujours trouvée peut être résolu par la soi-disant recherche de coût uniforme, dans laquelle le nœud avec le coût le plus bas parmi les nœuds triés par ordre croissant

```
RECHERCHE EN PROFONDEUR (nœud, objectif)

Si objectif atteint (nœud, objectif) retour ("solution trouvée")
NouveauNoeuds = Successeurs(Noeud)
Alors que NouveauNoeuds =
        Résultat = DEPTH-FIRST-SEARCH(First(NewNodes), Goal)
        Si Résultat = "Solution trouvée" Retour("Solution trouvée")
        NouveauxNoeuds = Reste(NouveauxNoeuds)
Retour ("Pas de solution")
```

Fig. 6.8 L'algorithme pour la recherche en profondeur d'abord. La fonction "First" retourne le premier élément d'une liste, et "Rest" le reste de la liste

La liste des nœuds est toujours étendue et les nouveaux nœuds triés. Ainsi, nous trouvons la solution optimale. Le problème de la mémoire n'est cependant pas encore résolu. Une solution à ce problème est fournie par la recherche en profondeur d'abord.

6.2.2 Recherche en profondeur d'abord

Dans la recherche en profondeur d'abord, seuls quelques nœuds sont stockés en mémoire à la fois. Après l'extension d'un nœud, seuls ses successeurs sont enregistrés et le premier nœud successeur est immédiatement étendu. Ainsi la recherche devient rapidement très profonde. Ce n'est que lorsqu'un nœud n'a pas de successeurs et que la recherche échoue à cette profondeur que le nœud ouvert suivant est étendu via un retour en arrière jusqu'à la dernière branche, et ainsi de suite. Nous pouvons le mieux percevoir cela dans l'élégant algorithme récursif de la Fig. 6.8 et dans l'arbre de recherche de la Fig. 6.9 à la page 92.

Analyse La recherche en profondeur d'abord nécessite beaucoup moins de mémoire que la recherche en largeur d'abord car au plus b nœuds sont enregistrés à chaque profondeur. Ainsi nous avons besoin de $b \cdot d$ cellules de mémoire.

Cependant, la recherche en profondeur d'abord n'est pas complète pour les arbres infiniment profonds car la recherche en profondeur d'abord s'exécute dans une boucle infinie lorsqu'il n'y a pas de solution dans la branche la plus à gauche. Par conséquent, la question de trouver la solution optimale est obsolète. A cause de la boucle infinie, aucune borne sur le temps de calcul ne peut être donnée. Dans le cas d'un arbre de recherche de profondeur finie avec une profondeur d, un total d'environ b^d nœuds est généré.
Ainsi, le temps de calcul croît, tout comme dans la recherche en largeur d'abord, de façon exponentielle avec la profondeur.

Nous pouvons rendre l'arbre de recherche fini en fixant une limite de profondeur. Maintenant, si aucune solution n'est trouvée dans l'arbre de recherche élagué, il peut néanmoins y avoir des solutions en dehors de la limite.
Ainsi la recherche devient incomplète. Il existe cependant des idées évidentes pour mener à bien la recherche.

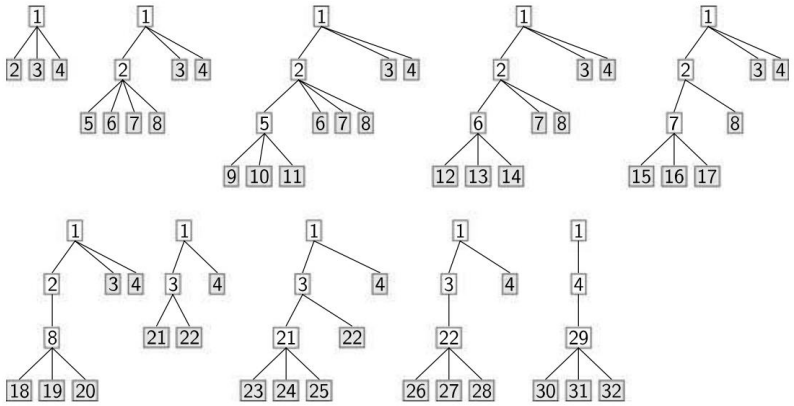


Fig. 6.9 Exécution de la recherche en profondeur d'abord. Tous les nœuds à la profondeur trois échouent et provoquent un retour en arrière. Les nœuds sont numérotés dans l'ordre où ils ont été générés

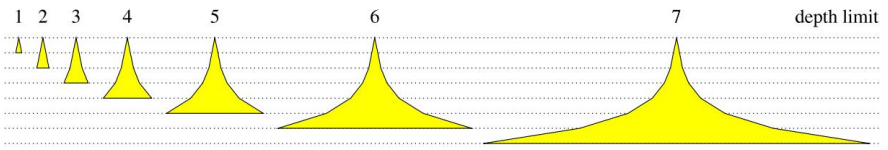


Fig. 6.10 Représentation schématique de l'évolution de l'arbre de recherche en approfondissement itératif avec des bornes de 1 à 7. La largeur de l'arbre correspond à un facteur de branchement de 2

6.2.3 Approfondissement itératif

Nous commençons la recherche en profondeur d'abord avec une limite de profondeur de 1. Si aucune solution n'est trouvée, nous augmentons la limite de 1 et commençons la recherche depuis le début, et ainsi de suite, comme le montre la Fig. 6.10 . Cette élévation itérative de la limite de profondeur est appelée approfondissement itératif.

Nous devons augmenter le programme de recherche en profondeur d'abord donné dans la Fig. 6.8 à la page 91 avec les deux paramètres supplémentaires « Profondeur » et « Limite ». "Depth" est augmenté de un lors de l'appel récursif, et la ligne d'en-tête de la boucle while est remplacée par "While NewNodes = And Depth < Limit". L'algorithme modifié est représenté dans la Fig. 6.11 à la page 93.

Analyse La mémoire requise est la même que pour la recherche en profondeur d'abord. On pourrait soutenir que le redémarrage répété de la recherche en profondeur d'abord à la profondeur zéro entraîne beaucoup de travail redondant. Pour les grands facteurs de branchement, ce n'est pas le cas. Nous montrons maintenant que la somme du nombre de nœuds de toutes les profondeurs jusqu'à l'avant-dernier dmax − 1 dans tous les arbres recherchés est beaucoup plus petite que le nombre de nœuds dans le dernier arbre recherché.

```
APPROFONDISSEMENT ITÉRATIF(Nœud, Objectif)

Limite de profondeur = 0
Répéter
    Résultat = DEPTHFIRSTSEARCH-B(Nœud, Objectif, 0, DepthFirstSearch)
    ProfondeurLimit = ProfondeurLimit + 1
Jusqu'au résultat = "Solution trouvée"
```

```
DEPTHFIRSTSEARCH-B(Nœud, Objectif, Profondeur, Limite)
Si objectif atteint (nœud, objectif) retour ("solution trouvée")
NouveauNoeuds = Successeurs(Noeud)
Tandis que NouveauNoeuds =     Et Profondeur < Limite
    Résultat =
        DEPTHFIRSTSEARCH-B(First(NewNodes), Goal, Depth + 1, Limit)
    Si Résultat = "Solution trouvée" Retour("Solution trouvée")
    NouveauxNoeuds = Reste(NouveauxNoeuds)
Retour ("Pas de solution")
```

Fig. 6.11 L'algorithme d'approfondissement itératif, qui appelle la recherche en profondeur d'abord légèrement modifiée avec une limite de profondeur (TIEFENSUCHE-B)

Soit Nb(d) le nombre de nœuds d'un arbre de recherche de facteur de branchement b et profondeur d et dmax la dernière profondeur recherchée. Le dernier arbre recherché contient

$$Nb(dmax) = \sum_{je=0}^{d_{maximum}} je\ b = \frac{b\ dmax+1 - 1}{b-1}$$

nœuds. Tous les arbres préalablement recherchés ensemble ont

$$Nb(j) = \sum_{j=1}^{dmax - 1} \frac{dmax - 1\ d+1 - 1}{b-1} = \frac{1}{b-1} \sum_{j=1}^{dmax - 1\ d+1\ b} - dmax + 1$$

$$= \frac{1}{b-1} \sum_{j=2}^{d_{maximum}} ré\ b - dmax + 1$$

$$= \frac{1}{b-1} \left(\frac{b\ dmax+1 -}{b-1} 1 - 1 - b - dmax + 1 \right)$$

$$\approx \frac{1}{b-1} \frac{b\ dmax+1 - 1}{b-1} = \frac{1}{b-1} Nb(dmax)$$

Tableau 6.1 Comparaison des algorithmes de recherche non informés. (*) signifie que la déclaration est uniquement true étant donné un coût d'action constant. ds est la profondeur maximale pour un arbre de recherche fini

	La largeur d'abord recherche	Coût uniforme recherche	Profondeur d'abord recherche	Itératif approfondissement
Complétude	Oui	Oui	Non	Oui
Solution optimale	Oui (*)	Oui	Non	Oui (*)
Temps de calcul	ré b	ré b	∞ ou b ds	b^d
Utilisation de la mémoire	b^d	b^d	bd	bd

nœuds. Pour b > 2 c'est inférieur au nombre Nb(dmax) de nœuds dans le dernier arbre. Pour b = 20, les premiers arbres dmax − 1 ne contiennent ensemble qu'environ = 1/19 de $\frac{1}{b-1}$ nombre de nœuds dans le dernier arbre. Le temps de calcul pour toutes les itérations en plus du dernier peut être ignoré.

Tout comme la recherche en largeur d'abord, cette méthode est complète et à coût constant pour toutes les actions, il trouve la solution la plus courte.

6.2.4 Comparaison

Les algorithmes de recherche décrits ont été mis côte à côte dans le tableau 6.1.

Nous pouvons clairement voir que l'approfondissement itératif est le gagnant de ce test car il obtient la meilleure note dans toutes les catégories. En fait, des quatre algorithmes présentés, c'est le seul pratiquement utilisable.

Nous avons en effet un gagnant pour ce test, bien que pour des applications réalistes, il ne réussit généralement pas. Même pour le puzzle 15, le grand frère du puzzle 8 (voir Exercice 6.4 page 110), il y a environ 2 × 1013 états différents. Pour non négligeable systèmes d'inférence, l'espace d'états est de plusieurs ordres de grandeur plus grand. Comme représenté sur la Secte. 6.1, toute la puissance de calcul du monde n'aidera pas beaucoup plus. Plutôt ce qu'il faut, c'est une recherche intelligente qui n'explore qu'une infime partie de la recherche l'espace et y trouve une solution.

6.3 Recherche heuristique

Les heuristiques sont des stratégies de résolution de problèmes qui, dans de nombreux cas, trouvent une solution plus rapidement que la recherche non informée. Cependant, cela n'est pas garanti. La recherche heuristique pourrait nécessitent beaucoup plus de temps et peuvent même aboutir à ce que la solution ne soit pas trouvée.

Nous, les humains, utilisons avec succès des processus heuristiques pour toutes sortes de choses. Quand acheter des légumes au supermarché, par exemple, nous jugeons les différentes options pour une livre de fraises en utilisant seulement quelques critères simples comme le prix, l'apparence, source de production et confiance dans le vendeur, puis nous décidons de la meilleure option par intuition. Il serait théoriquement préférable de soumettre les fraises à une analyse chimique avant de décider de les acheter. Par exemple, les fraises peuvent être empoisonnées. Si tel était le cas, l'analyse aurait valu la peine

```
RECHERCHEHEURISTIQUE(Début, Objectif)

NodeList = [Démarrer]
Bien que vrai
    Si NodeList =     Return("Pas de solution")
    Nœud = Premier (NodeList)
    NodeList = Rest(NodeList)
    If GoalReached(Node, Goal) Return("Solution found", Node)
    NodeList = SortIn(Successeurs(Node),NodeList)
```

Fig. 6.12 L'algorithme de recherche heuristique

inquiéter. Cependant, nous ne réalisons pas ce genre d'analyse car il y a une très forte probabilité que notre sélection heuristique réussisse et nous amène rapidement à notre objectif de manger des fraises savoureuses.

Les décisions heuristiques sont étroitement liées à la nécessité de prendre des décisions en temps réel avec des ressources limitées. En pratique, une bonne solution trouvée rapidement est préférée à une solution optimale, mais très coûteuse à dériver.

Une fonction d'évaluation heuristique f(s) pour les états est utilisée pour modéliser mathématiquement une heuristique. L'objectif est de trouver, avec peu d'effort, une solution au problème de recherche énoncé avec un coût total minimal. Veuillez noter qu'il existe une différence subtile entre l'effort pour trouver une solution et le coût total de cette solution. Par exemple, Google Maps peut prendre une demi-seconde d'efforts pour trouver un itinéraire de l'hôtel de ville de San Francisco à Tuolumne Meadows dans le parc national de Yosemite, mais le trajet de San Francisco à Tuolumne Meadows en voiture peut prendre quatre heures et un peu d'argent pour l'essence, etc. (coût total).

Ensuite, nous allons modifier l'algorithme de recherche en largeur d'abord en y ajoutant la fonction d'évaluation. Les nœuds actuellement ouverts ne sont plus développés de gauche à droite par ligne, mais plutôt en fonction de leur cote heuristique. Dans l'ensemble des nœuds ouverts, le nœud avec la note minimale est toujours développé en premier. Ceci est réalisé en évaluant immédiatement les nœuds au fur et à mesure qu'ils sont développés et en les triant dans la liste des nœuds ouverts. La liste peut alors contenir des nœuds de profondeurs différentes dans l'arbre.

Comme l'évaluation heuristique des états est très importante pour la recherche, nous différencierons désormais les états et leurs nœuds associés. Le noeud con

contient l'état et d'autres informations pertinentes pour la recherche, telles que sa profondeur dans l'arbre de recherche et l'évaluation heuristique de l'état. De ce fait, la fonction « Successeurs », qui génère les successeurs (enfants) d'un nœud, doit également calculer immédiatement pour ces nœuds successeurs leurs notes heuristiques en tant que composante de chaque nœud. Nous définissons l'algorithme de recherche général HEURISTICSEARCH dans la Fig. 6.12.

La liste de nœuds est initialisée avec les nœuds de départ. Ensuite, dans la boucle, le premier nœud de la liste est supprimé et testé pour savoir s'il s'agit d'un nœud de solution. Sinon, il sera élargi avec la fonction "Successeurs" et ses successeurs ajoutés à la liste

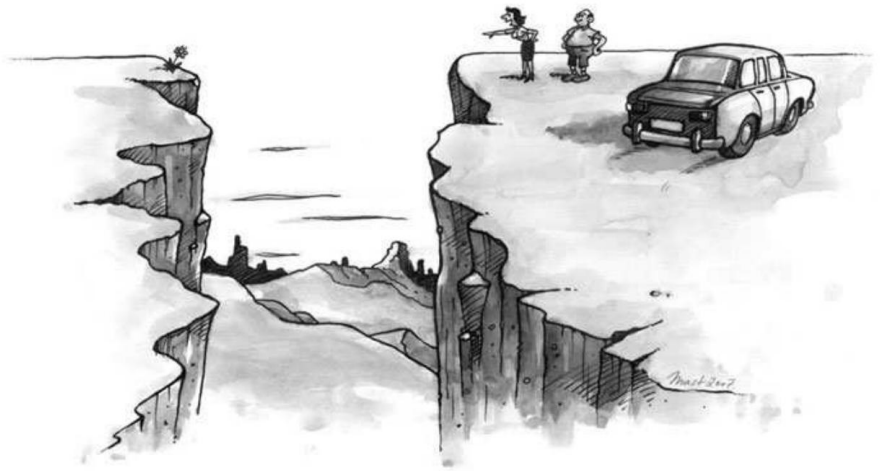


Fig. 6.13 Il : « Cher, pense au coût du carburant ! Je vais en cueillir un pour toi ailleurs. Elle: "Non, je veux celui-là là-bas!"

avec la fonction "SortIn". "SortIn(X,Y)" insère les éléments de la liste non triée X dans la liste triée par ordre croissant Y. La note heuristique est utilisée comme clé de tri.
Ainsi, il est garanti que le meilleur nœud (c'est-à-dire celui avec la valeur heuristique la plus faible) est toujours au début de la liste.3 Les

recherches en profondeur d'abord et en largeur d'abord sont aussi des cas particuliers de la fonction HEURISTICSEARCH. Nous pouvons facilement les générer en branchant la fonction d'évaluation appropriée (Exercice 6.11 à la page 111).

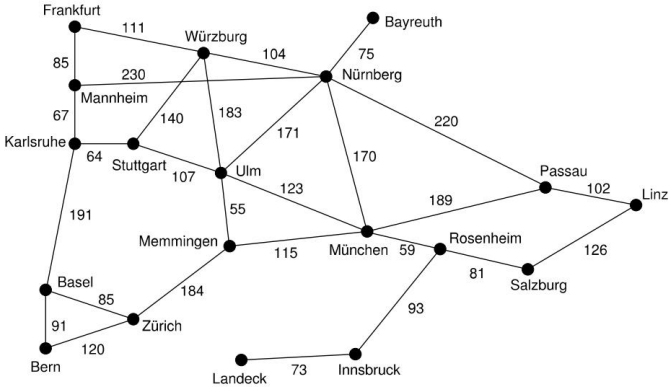
La meilleure heuristique serait une fonction qui calcule les coûts réels de chaque nœud à l'objectif. Pour ce faire, cependant, il faudrait parcourir tout l'espace de recherche, ce qui est exactement ce que l'heuristique est censée empêcher. Nous avons donc besoin d'une heuristique rapide et simple à calculer. Comment trouver une telle heuris tique ?

Une idée intéressante pour trouver une heuristique est la simplification du problème. La tâche d'origine est suffisamment simplifiée pour pouvoir être résolue avec un faible coût de calcul. Les coûts d'un état à l'objectif dans le problème simplifié servent alors d'estimation pour le problème réel (voir Fig. 6.13). Cette fonction d'estimation de coût est notée h.

6.3.1 Recherche gourmande

Il semble judicieux de choisir l'état avec la valeur h estimée la plus basse (c'est-à-dire celui avec le coût estimé le plus bas) dans la liste des états actuellement disponibles. Le

3Lors du tri dans un nouveau nœud de la liste des nœuds, il peut être avantageux de vérifier si le nœud est déjà disponible et, si c'est le cas, de supprimer le doublon.



Basel	204
Bayreuth	207
Bern	247
Frankfurt	215
Innsbruck	163
Karlsruhe	137
Landeck	143
Linz	318
München	120
Mannheim	164
Memmingen	47
Nürnberg	132
Passau	257
Rosenheim	168
Stuttgart	75
Salzburg	236
Würzburg	153
Zürich	157

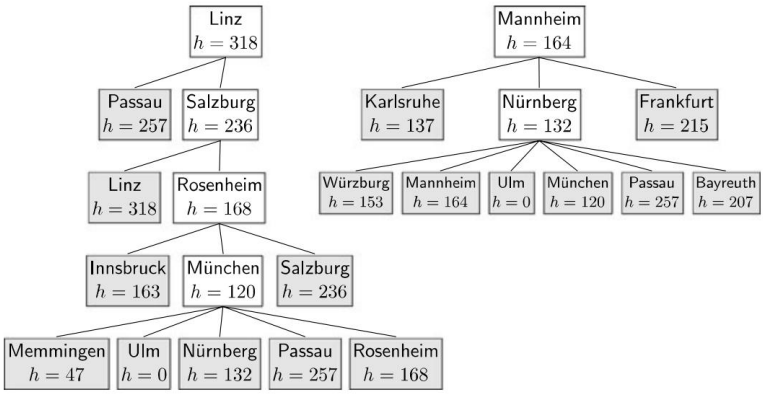
Fig. 6.14 Graphique des villes avec les distances de vol de toutes les villes à Ulm

l'estimation des coûts peut alors être utilisée directement comme fonction d'évaluation. Pour l'évaluation dans la fonction HEURISTICSEARCH nous posons f (s) = h(s). Cela se voit clairement dans l'exemple de planification de trajet (Exemple 6.2 à la page 88). Nous avons mis en place la tâche de trouver le chemin en ligne droite de ville en ville (c'est-à-dire la distance de vol) comme une simplification du problème. Au lieu de rechercher l'itinéraire optimal, nous déterminons d'abord à partir de chaque nœud un itinéraire avec une distance de vol minimale jusqu'au but. Nous choisissons Ulm comme destination. Ainsi, la fonction d'estimation des coûts devient

h(s) = distance de vol de la ville s à Ulm.

Les distances de vol de toutes les villes à Ulm sont données dans la Fig. 6.14 à côté du graphique.

L'arbre de recherche pour démarrer à Linz est représenté dans la Fig. 6.15 à la page 98 à gauche. On voit que l'arbre est très élancé. La recherche se termine donc rapidement. Malheureusement, cette recherche ne trouve pas toujours la solution optimale. Par exemple, cet algorithme ne parvient pas à trouver la solution optimale lors du démarrage à Mannheim (Fig. 6.15 à la page 98 à droite). Le chemin Mannheim–Nürnberg–Ulm a une longueur de 401 km. L'itinéraire Mannheim–Karlsruhe–Stuttgart–Ulm serait nettement plus court à 238 km. En observant le graphique, la cause de ce problème devient claire. Nuremberg est en fait un peu plus proche que Karlsruhe d'Ulm, mais la distance de Mannheim à Nürnberg est nettement plus grande que celle de Mannheim à Karlsruhe. L'heuristique ne fait qu'anticiper "avec avidité" vers l'objectif au lieu de prendre également en compte l'étirement qui a déjà été défini jusqu'au nœud actuel. C'est pourquoi nous lui donnons le nom de recherche gourmande.



Node	München	Innsbruck	Salzburg	Passau	Linz
heuristic rating	120	163	236	257	318

Fig. 6.15 Recherche gourmande : de Linz à Ulm (à gauche) et de Mannheim à Ulm (à droite). La structure de données de la liste des nœuds pour l'arbre de recherche de gauche, triée par la note du nœud avant l'expansion du nœud München est donnée

6.3.2 A -Recherche

Nous voulons maintenant prendre en compte les coûts qui se sont accumulés pendant la recherche jusqu'au nœud actuel s. On définit d'abord la fonction de coût

g(s) = Somme des coûts accumulés depuis le début jusqu'au nœud actuel,

puis ajouter à cela le coût estimé pour l'objectif et obtenir comme fonction d'évaluation heuristique

$$f(s) = g(s) + h(s).$$

Maintenant, nous ajoutons encore une autre exigence petite mais importante.

Définition 6.4 Une fonction heuristique d'estimation de coût h(s) qui ne surestime jamais le coût réel de l'état s au but est dite admissible.

La fonction HEURISTICSEARCH avec une fonction d'évaluation f (s) = g(s) + h(s) et une fonction heuristique admissible h est appelée A -algorithme. Ce fameux algorithme est complet et optimal. A trouve donc toujours la solution la plus courte pour chaque problème de recherche résoluble. Nous expliquerons et prouverons cela dans la discussion suivante.

Nous appliquons d'abord l' algorithme A à l'exemple. Nous recherchons le plus court chemin de Francfort à Ulm.

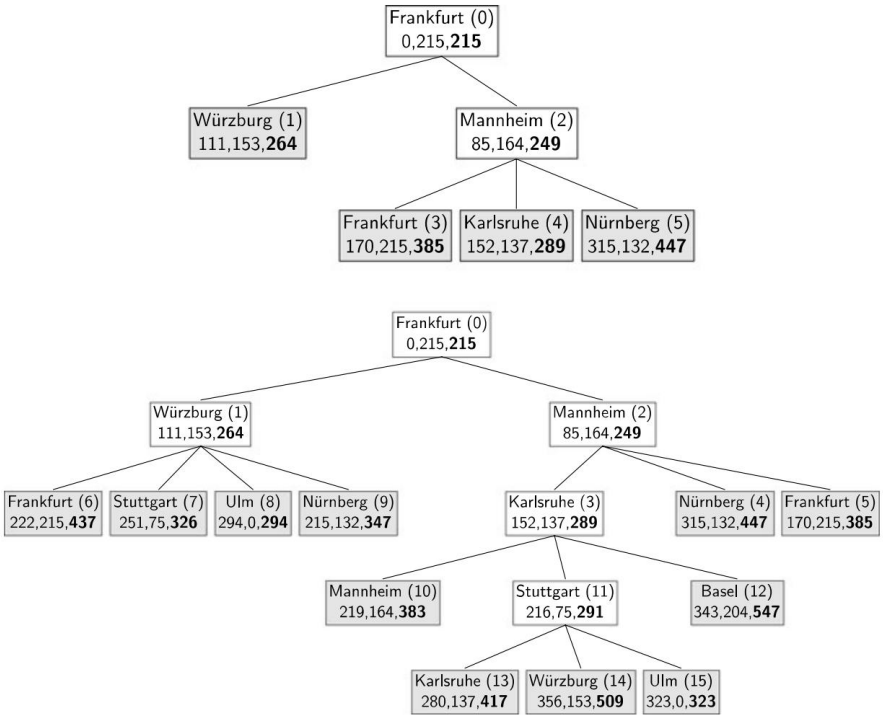


Fig. 6.16 Deux instantanés de l' arbre de recherche A pour l'itinéraire optimal de Francfort à Ulm. Dans les cases sous le nom de la ville s , nous indiquons g(s), h(s), f (s). Les nombres entre parenthèses après les noms de villes indiquent l'ordre dans lequel les nœuds ont été générés par la fonction "Successeur"

Dans la partie supérieure de la Fig. 6.16 , nous voyons que les successeurs de Mannheim sont générés avant les successeurs de Würzburg. La solution optimale Francfort-Wurtzbourg-Ulm est générée peu de temps après dans la huitième étape, mais elle n'est pas encore reconnue comme telle. Ainsi, l'algorithme ne se termine pas encore car le nœud Karlsruhe (3) a une meilleure valeur f (inférieure) et est donc en avance sur le nœud Ulm (8) en ligne. Ce n'est que lorsque toutes les valeurs de f sont supérieures ou égales à celle du nœud de solution Ulm (8) que nous nous sommes assurés d'avoir une solution optimale. Sinon, il pourrait y avoir une autre solution à moindre coût. Nous allons maintenant montrer que cela est vrai en général.

Théorème 6.2 L' algorithme A est optimal. Autrement dit, il trouve toujours la solution avec le coût total le plus bas si l'heuristique h est admissible.

Preuve Dans l' algorithme HEURISTICSEARCH , chaque nœud s nouvellement généré est trié par la fonction "SortIn" selon sa notation heuristique f (s). Le nœud avec la plus petite valeur de notation se trouve donc au début de la liste. Si le nœud I au

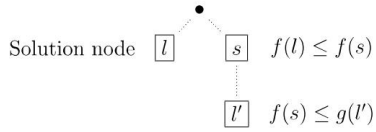


Fig. 6.17 Le premier nœud solution l trouvé par A n'a jamais un coût plus élevé qu'un autre nœud arbitraire l′

début de la liste est un nœud de solution, alors aucun autre nœud n'a une meilleure note heuristique. Pour tous les autres nœuds s il est vrai alors que f (l) ≤ f (s). Comme l'heuristique est admissible, aucune meilleure solution l ne peut être trouvée, même après expansion de tous les autres nœuds (voir Fig. 6.17). Ecrit formellement :

$$g(l) = g(l) + h(l) = f(l) \leq f(s) = g(s) + h(s) \leq g(l').$$

La première égalité tient car l est un nœud solution avec h(l) = 0. La seconde est la définition de f . La troisième (in)égalité tient parce que la liste des nœuds ouverts est triée par ordre croissant. La quatrième égalité est encore la définition de f . Enfin, la dernière (in)égalité est l'admissibilité de l'heuristique, qui ne surestime jamais le coût du nœud s à une solution arbitraire. Ainsi, il a été montré que g(l) ≤ g(l′), c'est-à-dire que la solution découverte l est optimale.

6.3.3 IDA -Recherche

La recherche A hérite d'une bizarrerie de la recherche en largeur d'abord. Il doit enregistrer de nombreux nœuds en mémoire, ce qui peut entraîner une utilisation très élevée de la mémoire. De plus, la liste des nœuds ouverts doit être triée. Ainsi l'insertion de nœuds dans la liste et la suppression de nœuds de la liste ne peuvent plus s'exécuter en temps constant, ce qui augmente légèrement la complexité de l'algorithme. Sur la base de l'algorithme de tri en tas, nous pouvons structurer la liste de nœuds comme un tas avec une complexité temporelle logarithmique pour l'insertion et la suppression de nœuds (voir [CLR90]).

Les deux problèmes peuvent être résolus - de la même manière que la recherche en largeur d'abord - par un approfondissement itératif. Nous travaillons avec une recherche en profondeur d'abord et élevons successivement la limite. Cependant, plutôt que de travailler avec une limite de profondeur, nous utilisons ici une limite pour l'évaluation heuristique f (s). Ce processus est appelé l' algorithme IDA .

6.3.4 Comparaison empirique des algorithmes de recherche

En A , ou (alternativement) IDA , nous avons un algorithme de recherche avec de nombreuses bonnes propriétés. Il est complet et optimal. Il peut donc être utilisé sans risque. La chose la plus importante, cependant, est qu'elle fonctionne avec des heuristiques, et peut donc réduire considérablement le temps de calcul nécessaire pour trouver une solution. Nous aimerions explorer cela empiriquement dans l'exemple de 8 puzzles.

Pour le 8-puzzle il y a deux heuristiques simples admissibles. L'heuristique h1 compte simplement le nombre de cases qui ne sont pas au bon endroit. Il est clair que cette heuristique est admissible. L'heuristique h2 mesure la distance de Manhattan. Pour chaque

Tableau 6.2 Comparaison du coût de calcul de la recherche non informée et de la recherche heuristique pour problèmes résolubles à 8 énigmes avec différentes profondeurs. Les mesures sont en pas et en secondes. Toutes les valeurs sont des moyennes sur plusieurs exécutions (voir la dernière colonne)

Profondeur	Étapes d'approfondissement		Algorithme A				Num.
	itératif	Temps [seconde]	Heuristique h1		Heuristique h2		court
			Étapes	Temps [sec]	Étapes	Temps [sec]	
2	20	0,003	3.0	0,0010	3.0	0,0010	dix
4	81	0,013	5.2	0,0015	5.0	0,0022	24
6	806	0,13	10.2	0,0034	8.3	0,0039	19
8	6455	1.0	17.3	0,0060	12.2	0,0063	14
dix	50512	7.9	48.1	0,018	22.1	0,011	15
12	486751	75,7	162.2	0,074	56,0	0,031	12
			IDA				
14	–	–	10079.2	2.6	855.6	0,25	16
16	–	–	69386.6	19.0	3806.5	1.3	13
18	–	–	708780.0	161.6	53941.5	14.1	4

mettez au carré les distances horizontales et verticales à l'emplacement de ce carré dans l'état d'objectif sont additionnés. Cette valeur est ensuite additionnée sur tous les carrés. Par exemple, le Manhattan distance des deux états

est calculé comme

$$h2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$$

L'admissibilité de la distance de Manhattan est également évidente (voir l'exercice 6.13 sur pages 111).

Les algorithmes décrits ont été implémentés dans Mathematica. Pour une comparaison en recherche non informée, l' algorithme A avec les deux heuristiques h1 et h2 et un approfondissement itératif a été appliqué à 132 problèmes de 8 puzzles générés aléatoirement. Le les valeurs moyennes du nombre d'étapes et du temps de calcul sont données dans le tableau 6.2. Nous voyons que l'heuristique réduit considérablement le coût de la recherche par rapport à la recherche non informée.

Si l'on compare l'approfondissement itératif à A avec h1 à la profondeur 12, par exemple, il devient évident que h1 réduit le nombre d'étapes d'un facteur d'environ 3 000, mais le temps de calcul par seulement un facteur de 1 023. Cela est dû au coût plus élevé par étape pour le calcul de l'heuristique.

Un examen plus approfondi révèle un saut dans le nombre de pas entre la profondeur 12 et profondeur 14 dans la colonne pour h1. Ce saut ne peut s'expliquer uniquement par la répétition travaux réalisés par IDA · Cela vient du fait que l'implémentation de l' algorithme A

supprime les doublons de nœuds identiques et réduit ainsi l'espace de recherche. Ce n'est pas possible avec IDA car il n'enregistre presque aucun nœud. Malgré cela, A ne peut plus rivaliser avec IDA au-delà de la profondeur 14 car le coût du tri dans les nouveaux nœuds augmente tellement le temps par étape.

Un calcul du facteur de branchement effectif selon (6.1) à la page 87 donne des valeurs d'environ 2,8 pour une recherche non informée. Ce nombre est cohérent avec la valeur de la Sect. 6.1. L'heuristique h1 réduit le facteur de branchement à des valeurs d'environ 1,5 et h2 à environ 1,3. Nous pouvons voir dans le tableau qu'une petite réduction du facteur de branchement de 1,5 à 1,3 nous donne un gros avantage en temps de calcul.

La recherche heuristique a donc une signification pratique importante car elle peut résoudre des problèmes qui sont hors de portée d'une recherche non informée.

6.3.5 Résumé

Parmi les différents algorithmes de recherche pour la recherche non informée, l'approfondissement itératif est le seul pratique car il est complet et peut se débrouiller avec très peu de mémoire.
Cependant, pour les problèmes de recherche combinatoire difficiles, même l'approfondissement itératif échoue généralement en raison de la taille de l'espace de recherche. La recherche heuristique aide ici par sa réduction du facteur de branchement effectif. L' algorithme IDA , comme l'approfondissement itératif, est complet et nécessite très peu de mémoire.

Les heuristiques ne donnent naturellement un avantage significatif que si l'heuristique est "bonne".
Lors de la résolution de problèmes de recherche difficiles, la tâche réelle du développeur consiste à concevoir des heuristiques qui réduisent considérablement le facteur de branchement effectif. Insecte. 6.5, nous traiterons ce problème et montrerons également comment les techniques d'apprentissage automatique peuvent être utilisées pour générer automatiquement des heuristiques.

En conclusion, il reste à noter que les heuristiques n'ont aucun avantage en termes de performances pour les problèmes insolubles car l'insolvabilité d'un problème ne peut être établie que lorsque l'arbre de recherche complet a été parcouru. Pour les problèmes décidables tels que le 8-puzzle, cela signifie que tout l'arbre de recherche doit être parcouru jusqu'à une profondeur maximale, qu'une heuristique soit utilisée ou non. L'heuristique est toujours un inconvénient dans ce cas, attribuable au coût de calcul de l'évaluation de l'heuristique. Cet inconvénient peut généralement être estimé par un facteur constant indépendant de la taille du problème. Pour les problèmes indécidables comme la preuve de formules PL1, l'arbre de recherche peut être infiniment profond. Cela signifie que, dans le cas insoluble, la recherche ne se termine potentiellement jamais. En résumé, nous pouvons dire ce qui suit : pour les problèmes solubles, les heuristiques réduisent souvent considérablement le temps de calcul, mais pour les problèmes insolubles, le coût peut même être plus élevé avec les heuristiques.

6.4 Jeux avec des adversaires

Les jeux pour deux joueurs, tels que les échecs, les dames, Othello et Go sont déterministes car chaque action (un mouvement) entraîne le même état enfant étant donné le même état parent. En revanche, le backgammon est non déterministe car son état enfant dépend

sur le résultat d'un lancer de dés. Ces jeux sont tous observables car chaque joueur connaît toujours l'état complet du jeu. De nombreux jeux de cartes, comme le poker par exemple, ne sont que partiellement observables car le joueur ne connaît pas les cartes des autres joueurs, ou n'en a qu'une connaissance partielle.

Les problèmes discutés jusqu'ici dans ce chapitre étaient déterministes et observables. Dans ce qui suit, nous examinerons des jeux qui, eux aussi, sont déterministes et observables. De plus, nous nous limiterons aux jeux à somme nulle. Ce sont des jeux dans lesquels chaque gain réalisé par un joueur signifie une perte de la même valeur pour l'adversaire. La somme du gain et de la perte est toujours égale à zéro. C'est le cas des jeux d'échecs, de dames, d'Othello et de Go, mentionnés ci-dessus.

6.4.1 Recherche Minimax

Le but de chaque joueur est de faire des mouvements optimaux qui aboutissent à la victoire. En principe, il est possible de construire un arbre de recherche et de le parcourir complètement (comme avec le 8-puzzle) pour une série de coups qui se traduira par la victoire. Cependant, il y a plusieurs particularités à surveiller :

1. Le facteur de branchement effectif aux échecs est d'environ 30 à 35. Dans un jeu typique avec 50 coups par joueur, l'arbre de recherche a plus de $30^{100} \approx 10^{148}$ nœuds feuilles .
   Ainsi, il n'y a aucune chance d'explorer complètement l'arbre de recherche. De plus, les échecs sont souvent joués avec une limite de temps. Du fait de cette exigence de temps réel, la recherche doit être limitée à une profondeur appropriée dans l'arbre, par exemple huit demi-coups.
   Étant donné que parmi les nœuds feuilles de cet arbre à profondeur limitée, il n'y a normalement pas de nœuds de solution (c'est-à-dire des nœuds qui terminent le jeu), une fonction d'évaluation heuristique B pour les positions sur le plateau est utilisée. Le niveau de jeu du programme dépend fortement de la qualité de cette fonction d'évaluation. C'est pourquoi nous traiterons plus en détail ce sujet dans la Sec. 6.5.
2. Dans la suite nous appellerons le joueur dont nous souhaitons optimiser le jeu Max, et son adversaire Min. Les mouvements de l'adversaire (Min) ne sont pas connus à l'avance, et donc l'arbre de recherche réel non plus. Ce problème peut être élégamment résolu en supposant que l'adversaire fait toujours le meilleur coup possible. Plus l'évaluation B(s) pour la position s est élevée, meilleure est la position s pour le joueur Max et plus elle est mauvaise pour son adversaire Min. Max essaie de maximiser l'évaluation de ses mouvements, tandis que Min effectue des mouvements qui se traduisent par une évaluation aussi faible que possible.

Un arbre de recherche avec quatre demi-mouvements et des évaluations de toutes les feuilles est donné à la Fig. 6.18 à la page 104. L'évaluation d'un nœud interne est dérivée récursivement comme le maximum ou le minimum de ses nœuds enfants, selon le niveau du nœud.

6.4.2 Élagage Alpha-Bêta

En basculant entre la maximisation et la minimisation, nous pouvons nous épargner beaucoup de travail dans certaines circonstances. L'élagage alpha-bêta fonctionne avec une recherche en profondeur d'abord jusqu'à une limite de profondeur prédéfinie. De cette manière, l'arbre de recherche est parcouru à partir de la gauche

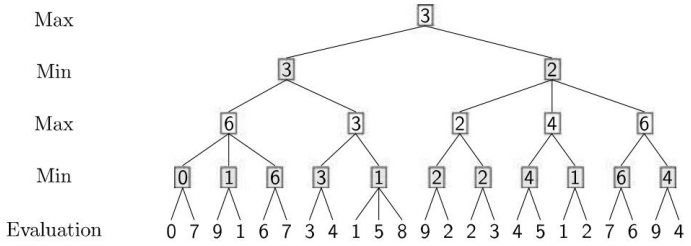


Fig. 6.18 Un arbre de jeu minimax avec anticipation de quatre demi-mouvements

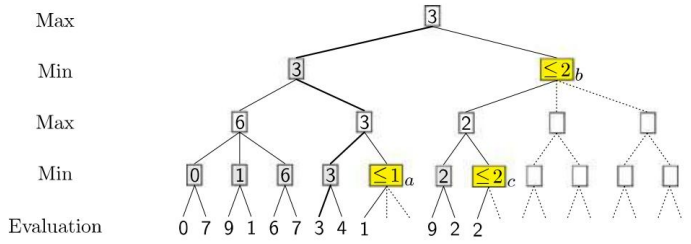


Fig. 6.19 Un arbre de jeu alpha-bêta avec anticipation de quatre demi-coups. Les parties en pointillés de l'arbre ne sont pas parcourues car elles n'ont aucun effet sur le résultat final

à droite. Comme dans la recherche minimax, dans les nœuds minimum, le minimum est généré à partir de la valeur minimale des nœuds successeurs et dans les nœuds maximum de même le maximum. Dans la Fig. 6.19, ce processus est décrit pour l'arbre de la Fig. 6.18. Au nœud marqué a, tous les autres successeurs peuvent être ignorés après que le premier enfant a été évalué comme la valeur 1 car le minimum est sûr d'être ≤1. Il pourrait même devenir encore plus petit, mais cela n'a pas d'importance puisque le maximum est déjà ≥3 un niveau au-dessus. Quelle que soit l'issue de l'évaluation des successeurs restants, le maximum conservera la valeur 3. De manière analogue, l'arbre sera coupé au nœud b. Puisque le premier enfant de b vaut 2, le minimum à générer pour b ne peut être qu'inférieur ou égal à 2. Mais le maximum au nœud racine est déjà sûr d'être ≥3.
Cela ne peut pas être modifié par des valeurs ≤2. Ainsi, les sous-arbres restants de b peuvent être élagués.

Le même raisonnement s'applique pour le nœud c. Cependant, le nœud maximal pertinent n'est pas le parent direct, mais le nœud racine. Cela peut être généralisé. • A chaque nœud feuille, l'évaluation est calculée. • Pour chaque nœud maximal, la plus grande valeur enfant actuelle est enregistrée dans α. • Pour chaque nœud minimum, la plus petite valeur enfant actuelle est enregistrée dans β. • Si à un nœud minimum k la valeur courante β ≤ α, alors la recherche sous k peut se terminer.

Ici α est la plus grande valeur d'un nœud maximum dans le chemin de la racine à k. • Si à un nœud maximum l la valeur courante α ≥ β, alors la recherche sous l peut se terminer.

Ici β est la plus petite valeur d'un nœud minimum dans le chemin de la racine à l.

```
ALPHABETAMAX(Nœud, α, β)

Si DepthLimitReached(Node) Return(Rating(Node))
NouveauNoeuds = Successeurs(Noeud)
Alors que NouveauNoeuds =
    α = Maximum(α, ALPHABETAMIN(First(NewNodes), α, β))
    Si α ≥ β Retour(β)
    NouveauxNoeuds = Reste(NouveauxNoeuds)
Retour(α)
```

```
ALPHABÉTAMINE(Nœud, α, β)

Si DepthLimitReached(Node) Return(Rating(Node))
NouveauNoeuds = Successeurs(Noeud)
Alors que NouveauNoeuds =
    β = Minimum(β, ALPHABETAMAX(First(NewNodes), α, β))
    Si β ≤ α Retour(α)
    NouveauxNoeuds = Reste(NouveauxNoeuds)
Retour(β)
```

Fig. 6.20 Algorithme de recherche alpha-bêta avec les deux fonctions ALPHABETAMIN et AL PHABETAMAX

L'algorithme donné à la Fig. 6.20 est une extension de la recherche en profondeur d'abord avec deux fonctions qui sont appelées en alternance. Il utilise les valeurs définies ci-dessus pour α et β.

Complexité Le temps de calcul économisé par l'élagage alpha-bêta dépend fortement de l'ordre dans lequel les nœuds enfants sont traversés. Dans le pire des cas, l'élagage alpha-bêta n'offre aucun avantage. Pour un facteur de branchement b constant, le nombre nd de nœuds feuilles à évaluer à la profondeur d est égal à

$$nd = b^{d}.$$

Dans le meilleur des cas, lorsque les successeurs des nœuds maximaux sont triés par ordre décroissant et les successeurs des nœuds minimaux sont triés par ordre croissant, le facteur de branchement effectif est réduit à $\sqrt{b}$. Aux échecs, cela signifie une réduction substantielle du facteur de branchement effectif de 35 à environ 6. Alors seulement

$$nd = \sqrt{b}^{d} = b^{d/2}$$

nœuds feuilles seraient créés. Cela signifie que la limite de profondeur et donc aussi l'horizon de recherche sont doublés avec l'élagage alpha-bêta. Cependant, cela n'est vrai que dans le cas de successeurs triés de manière optimale car les notes des nœuds enfants sont inconnues au moment de leur création. Si les nœuds enfants sont triés de manière aléatoire, le facteur de branchement est réduit à b 3/4 et le nombre de nœuds feuille à

$$nd = b^{\frac{3}{4}j}.$$

Avec la même puissance de calcul, un ordinateur d'échecs utilisant l'élagage alpha-bêta peut, par exemple, calculer huit demi-coups en avant au lieu de six, avec un facteur de branchement effectif d'environ 14. Une analyse approfondie avec une dérivation de ces paramètres peut être trouvée dans [Pois84].

Pour doubler la profondeur de recherche comme mentionné ci-dessus, il faudrait que les nœuds enfants soient ordonnés de manière optimale, ce qui n'est pas le cas en pratique. Sinon la recherche serait inutile. Avec une astuce simple, nous pouvons obtenir un ordre de nœuds relativement bon. Nous relions l'élagage alpha-bêta à l'approfondissement itératif au-delà de la limite de profondeur. Ainsi, à chaque nouvelle limite de profondeur, nous pouvons accéder aux notes de tous les nœuds des niveaux précédents et ordonner les successeurs à chaque branche. On atteint ainsi un facteur de branchement effectif d'environ 7 à 8, ce qui n'est pas loin de l'optimum théorique de $\sqrt{35}$ [Nil98].

6.4.3 Jeux non déterministes

La recherche Minimax peut être généralisée à tous les jeux avec des actions non déterministes, comme le backgammon. Chaque joueur lance avant son coup, qui est influencé par le résultat du jet de dés. Dans l'arbre du jeu, il y a donc maintenant trois types de niveaux dans la séquence

Max, dés, Min, dés, . . . ,

où chaque nœud de jet de dés se ramifie de six façons. Parce que nous ne pouvons pas prédire la valeur du dé, nous faisons la moyenne des valeurs de tous les lancers et effectuons la recherche comme décrit avec les valeurs moyennes de [RN10].

6.5 Fonctions d'évaluation heuristiques

Comment trouver une bonne fonction d'évaluation heuristique pour la tâche de recherche ?
Ici, il y a fondamentalement deux approches. La voie classique utilise les connaissances d'experts humains. L'ingénieur des connaissances se voit confier la tâche généralement difficile de formaliser les connaissances implicites de l'expert sous la forme d'un programme informatique.
Nous voulons maintenant montrer comment ce processus peut être simplifié dans l'exemple du programme d'échecs.

Dans un premier temps, les experts sont interrogés sur les facteurs les plus importants dans le choix d'un déménagement. Ensuite, on tente de quantifier ces facteurs. Nous obtenons une liste de caractéristiques ou d'attributs pertinents. Ceux-ci sont ensuite (dans le cas le plus simple) combinés en

une fonction d'évaluation linéaire B(s) pour les postes, qui pourrait ressembler à :

$$B(s) = a1 \cdot \text{matériel} + a2 \cdot \text{pion\_structure} + a3 \cdot \text{roi\_sécurité} + a4 \cdot \text{Knight\_in\_center} + a5 \cdot \text{Bishop\_diagonal\_coverage} + \cdots , \quad (6.3)$$

où le « matériel » est de loin la caractéristique la plus importante et est calculé par

$$\text{matériel} = \text{matériel(propre\_équipe)} - \text{matériel(adversaire)}$$

avec

$$\text{matériel(équipe)} = \text{num\_pions(équipe)} \cdot 100 + \text{num\_knights(équipe)} \cdot 300 + \text{num\_bishops(équipe)} \cdot 300 + \text{num\_rooks(équipe)} \cdot 500 + \text{num\_queens(équipe)} \cdot 900$$

Presque tous les programmes d'échecs font une évaluation similaire pour le matériel. Cependant, il existe de grandes différences pour toutes les autres fonctionnalités, que nous n'aborderons pas ici [Fra05, Lar00].

Dans l'étape suivante, les poids ai de toutes les caractéristiques doivent être déterminés. Ceux-ci sont définis intuitivement après discussion avec des experts, puis modifiés après chaque partie en fonction de l'expérience positive et négative. Le fait que ce processus d'optimisation soit très coûteux et de plus que la combinaison linéaire des fonctionnalités soit très limitée suggère l'utilisation de l'apprentissage automatique.

6.5.1 Apprentissage des heuristiques

Nous voulons maintenant optimiser automatiquement les poids ai de la fonction d'évaluation B(s) de (6.3). Dans cette approche, l'expert n'est interrogé que sur les caractéristiques pertinentes f1(s), . . . , fn(s) pour l'état du jeu s. Ensuite, un processus d'apprentissage automatique est utilisé dans le but de trouver une fonction d'évaluation aussi proche que possible de l'optimale. Nous commençons avec une fonction d'évaluation initiale prédéfinie (déterminée par le processus d'apprentissage), puis laissons le programme d'échecs jouer. À la fin de la partie, une note est dérivée du résultat (victoire, défaite ou match nul). Sur la base de cette note, la fonction d'évaluation est modifiée dans le but de faire moins d'erreurs la prochaine fois. En principe, la même chose qui est faite par le développeur est maintenant prise en charge automatiquement par le processus d'apprentissage.

Aussi facile que cela puisse paraître, c'est très difficile dans la pratique. Un problème central lié à l'amélioration de la cote de position en fonction des matchs gagnés ou perdus est connu aujourd'hui sous le nom de problème d'attribution de crédit . Nous avons en fait une note à la fin du jeu, mais pas de notes pour les mouvements individuels. Ainsi, l'agent effectue de nombreuses actions mais ne reçoit aucun retour positif ou négatif jusqu'à la toute fin. Comment doit-elle alors attribuer ce retour d'expérience aux nombreuses actions entreprises dans le passé ? Et comment devrait-elle améliorer ses actions dans ce cas ? Le jeune domaine passionnant de l'apprentissage par renforcement s'intéresse à ces questions (voir section 10).

La plupart des meilleurs ordinateurs d'échecs au monde fonctionnent encore aujourd'hui sans techniques d'apprentissage automatique. Il y a deux raisons à cela. D'une part les algorithmes d'apprentissage par renforcement développés jusqu'à présent nécessitent beaucoup de temps de calcul compte tenu de grands espaces d'états. D'un autre côté, l'heuristique créée manuellement des ordinateurs d'échecs hautes performances est déjà fortement optimisée. Cela signifie que seul un très bon système d'apprentissage peut conduire à des améliorations. Dans les dix prochaines années, le temps viendra vraisemblablement où un ordinateur d'apprentissage deviendra champion du monde.

6.6 État de l'art

Pour l'évaluation de la qualité des processus de recherche heuristique, je voudrais répéter La définition d'Elaine Rich [Ric83] :

> L'intelligence artificielle est l'étude de la façon de faire faire aux ordinateurs des choses dans lesquelles, pour le moment, les gens sont meilleurs.

Il n'y a guère de meilleur test pour décider si un programme informatique est intelligent que la comparaison directe de l'ordinateur et de l'humain dans un jeu comme les échecs, les dames, le backgammon ou le go.

En 1950, Claude Shannon, Konrad Zuse et John von Neumann ont introduit les premiers programmes d'échecs, qui, cependant, ne pouvaient pas être mis en œuvre ou prendraient beaucoup de temps à mettre en œuvre. Quelques années plus tard, en 1955, Arthur Samuel écrivit un programme qui jouait aux dames et pouvait améliorer ses propres paramètres grâce à un simple processus d'apprentissage. Pour ce faire, il a utilisé le premier ordinateur à logique programmable, l'IBM 701. Par rapport aux ordinateurs d'échecs d'aujourd'hui, cependant, il avait accès à un grand nombre de parties archivées, pour lesquelles chaque coup individuel avait été évalué par des experts. Ainsi, le programme a amélioré sa fonction d'évaluation. Pour obtenir de nouvelles améliorations, Samuel a fait jouer son programme contre lui-même. Il a résolu le problème d'attribution de crédit d'une manière simple. Pour chaque position individuelle au cours d'un jeu, il compare l'évaluation par la fonction B(s) avec celle calculée par l'élagage alpha bêta et modifie B(s) en conséquence. En 1961, son programme de dames a battu le quatrième meilleur joueur de dames aux États-Unis. Avec ce travail révolutionnaire, Samuel avait sûrement près de 30 ans d'avance sur son temps.

Ce n'est qu'au début des années 90, avec l'émergence de l'apprentissage par renforcement, que Gerald Tersauro a créé un programme d'apprentissage du backgammon nommé TD-Gammon, qui jouait au niveau des champions du monde (voir section 10).

Aujourd'hui, plusieurs programmes d'échecs existent qui jouent au niveau grand maître, et certains sont vendus dans le commerce pour PC. La percée a eu lieu en 1997, lorsque Deep Blue d'IBM a battu le champion du monde d'échecs Gary Kasparov avec un score de 3,5 parties contre 2,5. Deep Blue pouvait en moyenne calculer 12 demi-mouvements en avant avec l'élagage alpha-bêta et l'évaluation heuristique de la position.

L'un des ordinateurs d'échecs les plus puissants à ce jour est Hydra, un ordinateur parallèle appartenant à une société des Émirats arabes unis. Le logiciel a été développé par les scientifiques Christian Donninger (Autriche) et Ulf Lorenz (Allemagne), ainsi que par le grand champion d'échecs allemand Christopher Lutz. Hydra utilise 64 Xeon parallèles

processeurs avec une puissance de calcul d'environ 3 GHz et 1 Go de mémoire chacun. Pour la fonction d'évaluation de position, chaque processeur dispose d'un coprocesseur FPGA (réseau de portes programmables par l'utilisateur). Ainsi, il devient possible d'évaluer 200 millions de positions par seconde même avec une fonction d'évaluation coûteuse.

Avec cette technologie, Hydra peut calculer en moyenne environ 18 coups d'avance.
Dans des situations spéciales et critiques, l'horizon de recherche peut même être étendu à 40 demi-mouvements. Il est clair que ce genre d'horizon dépasse ce que même les grands champions peuvent faire, car Hydra fait souvent des mouvements que les grands champions ne peuvent pas comprendre, mais qui finissent par mener à la victoire. En 2005, Hydra a battu le septième grand maître Michael Adams avec 5,5 à 0,5 matchs.

Hydra utilise peu de connaissances spéciales sur les échecs dans les manuels scolaires, plutôt une recherche alpha-bêta avec des heuristiques relativement générales et bien connues et une bonne évaluation de position codée à la main. En particulier, Hydra n'est pas capable d'apprendre. Des améliorations sont effectuées entre les jeux par les développeurs. Ainsi, l'ordinateur est toujours soulagé d'avoir à apprendre. Hydra n'a pas non plus d'algorithmes de planification spéciaux. Le fait qu'Hydra fonctionne sans apprentissage est un indice que, malgré de nombreuses avancées, il existe toujours un besoin de recherche en apprentissage automatique. Comme mentionné ci-dessus, nous entrerons plus en détail à ce sujet dans la section. 10 et Chap. 8.

En 2009, le système Pocket Fritz 4, fonctionnant sur un PDA, a remporté le tournoi d'échecs Copa Mercosur à Buenos Aires avec neuf victoires et un match nul contre 10 excellents joueurs d'échecs humains, dont trois grands maîtres. Même si peu d'informations sur la structure interne du logiciel sont disponibles, cette machine d'échecs représente une tendance à s'éloigner de la puissance de calcul brute vers plus d'intelligence. Cette machine joue au niveau grand maître et est comparable, sinon meilleure, à Hydra.
Selon le développeur de Pocket Fritz Stanislav Tsukrov [Wik10], Pocket Fritz avec son moteur de recherche d'échecs HIARCS 13 recherche moins de 20 000 positions par seconde, ce qui est plus lent qu'Hydra d'un facteur d'environ 10 000. Cela conduit à la conclusion que HIARCS 13 utilise définitivement une meilleure heuristique pour diminuer le facteur de branchement effectif qu'Hydra et peut donc bien être qualifié de plus intelligent qu'Hydra. Soit dit en passant, HIARCS est un raccourci pour Higher Intelligence Auto Response Chess System.

Même si bientôt aucun humain n'aura une chance contre les meilleurs ordinateurs d'échecs, il reste encore de nombreux défis pour l'IA. Allez, par exemple. Dans ce vieux jeu japonais joué sur un plateau carré avec 361 cases, 181 pierres blanches et 180 pierres noires, le facteur de branchement effectif est d'environ 300. Après quatre demi-coups, il y a déjà environ 8×109 positions . Tous les processus de recherche d'arbres de jeu classiques connus n'ont aucune chance contre de bons joueurs humains de Go à ce niveau de complexité. Les experts s'accordent à dire que des systèmes "vraiment intelligents" sont nécessaires ici. L'énumération combinatoire de toutes les possibilités est la mauvaise méthode. Ce qu'il faut bien plus, ce sont des processus capables de reconnaître les modèles sur le tableau, de suivre les développements à long terme et de prendre rapidement des décisions « intuitives ». Tout comme dans la reconnaissance d'objets dans des images complexes, les humains ont encore des kilomètres d'avance sur les programmes informatiques. Nous traitons l'image dans son ensemble de façon très parallèle, alors que l'ordinateur traite les millions de pixels les uns après les autres et a du mal à voir l'essentiel dans la plénitude des pixels. Le programme "Les multiples visages du go" peut reconnaître 1 100 schémas différents et connaît 200 stratégies de jeu. Mais tous les programmes Go ont encore de sérieux problèmes pour reconnaître si un groupe de pierres est mort ou vivant, ou où les regrouper entre les deux.

6.7 Des exercices

Exercice 6.1 (a)

Démontrer le théorème 6.1 page 88, en d'autres termes, prouver que pour un arbre avec un grand facteur de branchement constant b, presque tous les nœuds sont au dernier niveau à la profondeur d. (b) Montrer que cela n'est pas toujours vrai lorsque le facteur de branchement effectif est grand et non constant.

Exercice 6.2 (a)

Calculez le facteur de ramification moyen pour le puzzle en 8 sans vérifier les cycles. Le facteur de ramification moyen est le facteur de ramification qu'aurait un arbre avec un nombre égal de nœuds au dernier niveau, un facteur de ramification constant et une profondeur égale. (b) Calculez le facteur de branchement moyen pour le 8-puzzle pour une recherche non informée tout en évitant les cycles de longueur 2.

Exercice 6.3 (a)

Quelle est la différence entre le facteur de branchement moyen et effectif (Définition 6.2 à la page 87) ? (b) Pourquoi le facteur de branchement effectif est-il mieux adapté à l'analyse et à la comparaison du temps de calcul des algorithmes de recherche que le facteur de branchement moyen ? (c) Montrer que pour un arbre fortement ramifié à n nœuds et de profondeur d, le facteur de branchement effectif $\bar{b}$ est approximativement égal au facteur de branchement moyen et donc égal à $\sqrt[d]{n}$.

Exercice 6.4 (a)

Calculez la taille de l'espace d'états pour le 8-puzzle, pour le 3-puzzle analogue (matrice 2 × 2), ainsi que pour le 15-puzzle (4 × 4-matrice). (b) Prouver que le graphe d'état composé des états (nœuds) et des actions (arêtes) pour le 3-puzzle tombe dans deux sous-graphes connectés, entre lesquels il n'y a pas de connexions.

Exercice 6.5 Avec la recherche en largeur d'abord du 8-puzzle, trouvez un chemin (manuellement) à partir de le nœud de départ  au nœud de but  .

Exercice 6.6

(a) Programmez la recherche en largeur d'abord, la recherche en profondeur d'abord et l'approfondissement itératif dans la langue de votre choix et testez-les sur l'exemple de 8 puzzles. (b) Pourquoi est-il peu logique d'utiliser la recherche en profondeur d'abord sur le 8-puzzle ?

Exercice 6.7 (a)

Montrer que la recherche en largeur à coût constant pour toutes les actions est garantie pour trouver la solution la plus courte.

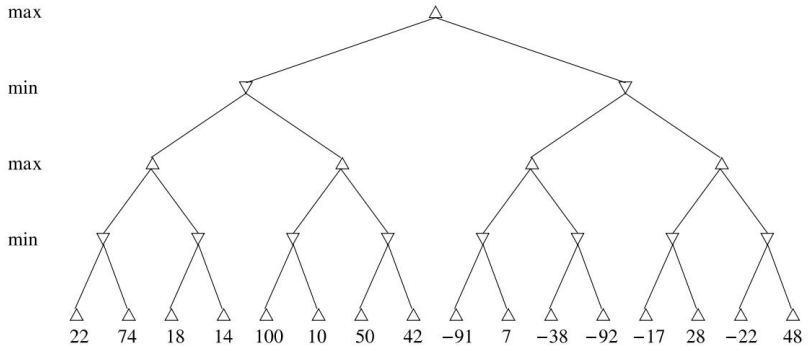


Fig. 6.21 Arbre de recherche Minimax

(b) Montrer que ce n'est pas le cas pour des coûts variables.

Exercice 6.8 En utilisant A recherchez le 8-puzzle, recherchez (manuellement) un chemin depuis le nœud de départ

1		3
4	2	6
7	5	8

au nœud de but

1	2	3
4	5	6
7	8	

(a) en utilisant l'heuristique h1 (Section 6.3.4). (b) en utilisant l'heuristique h2 (Section 6.3.4).

Exercice 6.9 Construisez l' arbre de recherche A pour le graphe de villes de la Fig. 6.14 à la page 97 et utilisez la distance de vol jusqu'à Ulm comme heuristique. Commencez à Berne avec Ulm comme destination. Veillez à ce que chaque ville n'apparaisse qu'une seule fois par chemin.

Exercice 6.10 Programme A rechercher dans le langage de programmation de votre choix en utilisant les heuristiques h1 et h2 et testez-les sur l'exemple de 8 puzzles.

Exercice 6.11 Donner une fonction d'évaluation heuristique pour les états avec lesquels HEURISTICSEARCH peut être implémenté comme une recherche en profondeur d'abord, et un pour une implémentation de recherche en largeur d'abord.

Exercice 6.12 Quelle est la relation entre l'image du couple au canyon de la Fig. 6.13 page 96 et l'heuristique admissible ?

Exercice 6.13 Montrer que les heuristiques h1 et h2 pour le 8-puzzle de la Sect. 6.3.4 sont recevables.

Exercice 6.14 (a)

L'arbre de recherche pour un jeu à deux joueurs est donné à la Fig. 6.21 avec les notes de tous les nœuds feuilles. Utilisez la recherche minimax avec élagage α-β de gauche à droite. Rayez tous les nœuds qui ne sont pas visités et donnez la note résultante optimale pour chaque nœud interne. Marquez le chemin choisi. (b) Testez-vous en utilisant l'applet de P. Winston [Win].

Pour résoudre ce problème, diverses formes de logique non monotone ont été développées, qui permettent de retirer des connaissances (formules) de la base de connaissances.
Sous le nom de logique par défaut, des logiques ont été développées qui permettent d'attribuer aux objets des attributs valables tant qu'aucune autre règle n'est disponible. Dans l'exemple de Tweety, la règle que les oiseaux peuvent voler serait une telle règle par défaut. Malgré de grands efforts, ces logiques n'ont pas abouti à l'heure actuelle, en raison de problèmes sémantiques et pratiques.

La monotonie peut être particulièrement gênante dans les problèmes de planification complexes dans lesquels le monde peut changer. Si par exemple une maison bleue est peinte en rouge, alors après elle est rouge. Une base de connaissances telle que

couleur(maison, bleu)
peinture(maison,rouge)
peinture(x, y) couleur(x, y)

conduit à la conclusion qu'après peinture, la maison est rouge et bleue. Le problème qui se pose ici dans la planification est connu sous le nom de problème de cadre. Une solution pour cela est le calcul de situation présenté dans la Sect. 5.6.

Une approche intéressante pour modéliser des problèmes tels que l'exemple de Tweety est la théorie des probabilités. L'affirmation "tous les oiseaux peuvent voler" est fausse. Une déclaration quelque chose comme "presque tous les oiseaux peuvent voler" est correcte. Cette affirmation devient plus exacte si nous donnons une probabilité pour « les oiseaux peuvent voler ». Cela conduit à la logique probabiliste, qui représente aujourd'hui un sous-domaine important de l'IA et un outil important pour modéliser l'incertitude (voir chap. 7).

4.4 Incertitude de modélisation

La logique à deux valeurs ne peut et ne devrait modéliser que des circonstances dans lesquelles il y a vrai, faux et aucune autre valeur de vérité. Pour de nombreuses tâches du raisonnement quotidien, la logique à deux valeurs n'est donc pas assez expressive. La règle

oiseau(x) voler(x)

est vrai pour presque tous les oiseaux, mais pour certains c'est faux. Comme nous l'avons déjà mentionné, travailler avec des probabilités permet une formulation exacte de l'incertitude. L'affirmation « 99 % de tous les oiseaux peuvent voler » peut être formalisée par l'expression

P (oiseau(x) voler(x)) = 0,99.

Au Chap. 7 nous verrons qu'ici il vaut mieux travailler avec des probabilités conditionnelles telles que

P (mouche|oiseau) = 0,99.

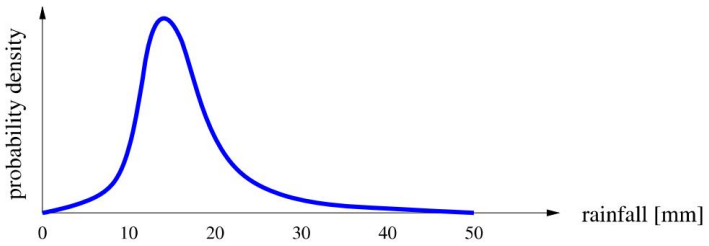


Fig. 4.3 Densité de probabilité de la pluie variable continue

Avec l'aide des réseaux bayésiens, des applications complexes avec de nombreuses variables peuvent également être modélisée.

Un modèle différent est nécessaire pour l'énoncé « Il fait beau ». Ici c'est souvent n'a aucun sens de parler en termes de vrai et de faux. La variable weather_is_nice ne doit pas être modélisé comme binaire, plutôt en continu avec des valeurs, par exemple, dans l'intervalle [0, 1]. weather_is_nice = 0.7 signifie alors "Il fait assez beau". La logique floue, qui sera également présentée au Chap. 7, a été développé pour ce type de variable continue (floue).

La théorie des probabilités offre également la possibilité de faire des déclarations sur la probabilité de variables continues. Une déclaration dans le bulletin météo telle que "Il y a une forte probabilité qu'il y ait de la pluie » pourrait par exemple être exactement formulée comme une densité de probabilité de la forme

$$P(\text{précipitations} = X) = Y$$

et représenté graphiquement quelque chose comme dans la Fig. 4.3.

Cette représentation très générale et même visualisable des deux types d'incertitude dont nous venons de parler, jointe à la statistique inductive et à la théorie des réseaux bayésiens, permet, en principe, de répondre à des probabilistes arbitraires. requêtes.

La théorie des probabilités ainsi que la logique floue ne sont pas directement comparables au prédicat logique car ils ne permettent pas de variables ou de quantificateurs. Ils peuvent ainsi être vus comme extensions de la logique propositionnelle comme indiqué dans le tableau suivant.

Formalisme	Nombre de valeurs de vérité	Probabilités exprimable
Logique propositionnelle	2	–
Logique floue	∞	–
Logique probabiliste discrète	n	Oui
Logique probabiliste continue	∞	Oui

4.5 Des exercices

Exercice 4.1

(a) Avec l'argument (faux) suivant, on pourrait prétendre que PL1 est décidable : On prend un calcul de preuve complet pour PL1. Avec elle, nous pouvons trouver une preuve pour toute formule vraie en temps fini. Pour toute autre formule φ , je procède comme suit : j'applique le calcul à ¬φ et je montre que ¬φ est vrai. Donc φ est faux. Ainsi, je peux prouver ou réfuter chaque formule de PL1. Trouvez l'erreur dans l'argument et modifiez-la pour qu'elle devienne

correcte. (b) Construire un processus de décision pour l'ensemble des formules vraies et insatisfaisantes dans PL1.

Exercice 4.2

(a) Compte tenu de l'énoncé "Il y a un barbier qui rase toute personne qui ne se rase pas." Considérez si ce barbier se rase. (b) Soit M = {x|x / x}. Décrivez

cet ensemble et demandez-vous si M se contient lui-même.

Exercice 4.3 Utilisez un démonstrateur de théorème automatisé (par exemple E [Sch02]) et appliquez-le aux cinq axiomatisations différentes de l'exemple Tweety de la Sect. 4.3. Validez les instructions de l'exemple.

Programmation logique avec PROLOG

5

Comparé aux langages de programmation classiques tels que C ou Pascal, Logic rend possible d'exprimer des relations de manière élégante, compacte et déclarative. automatique les démonstrateurs de théorèmes sont même capables de décider si une base de connaissances logiquement implique une requête. Le calcul de preuve et les connaissances stockées dans la base de connaissances sont strictement séparés. Une formule décrite sous forme normale de clause peut être utilisée comme entrée données pour tout démonstrateur de théorème, indépendamment du calcul de preuve utilisé. C'est de grande valeur pour le raisonnement et la représentation des connaissances.

Si l'on souhaite implémenter des algorithmes, qui ont inévitablement des composantes procédurales, une description purement déclarative est souvent insuffisante. Robert Kowalski, un des pionniers de la programmation logique, a fait valoir ce point avec la formule

Algorithme = Logique + Contrôle.

Cette idée a été concrétisée dans le langage PROLOG. PROLOG est utilisé dans de nombreux projets, principalement en IA et en linguistique computationnelle. Nous allons maintenant donner une courte introduction à ce langage, présenter les concepts les plus importants, montrer ses points forts et comparez-le avec d'autres langages de programmation et démonstrateurs de théorèmes. Ceux qui recherchent un cours de programmation complet sont dirigés vers des manuels tels que [Bra11, CM94] et les manuels [Wie04, Dia04].

La syntaxe du langage PROLOG n'autorise que les clauses Horn. Notation logique et la syntaxe de PROLOG sont juxtaposées dans le tableau suivant :

PL1 / clause forme normale	PROLOG	Description
(¬A1 ··· ¬Am B)	B :- A1, ... , Am.	Règle
(A1 ··· Am) B	B :- A1, ... , Am.	Règle
UN	UN.	Fait
(¬A1 ··· ¬Am)	?- A1, ... , Am.	Requête
¬(A1 ··· Suis)	?- A1, ... , Am.	Requête

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique,
DOI 10.1007/978-0-85729-299-5_5,

```
1 enfant (oscar, karen, franc). 2 enfants
(marie, karen, franc). 3
enfant(eve,anne,oscar). 4 enfants
(henri, anne, oscar). 5
enfants(Isolde,Anne,Oscar). 6 enfants
(clyde, mary, oscarb). 7 8 enfant(X,Z,Y) :-

enfant(X,Y,Z). 9

10 descendant(X,Y) :- enfant(X,Y,Z). 11 descendant(X,Y) :-
enfant(X,U,V), descendant(U,Y).
```

Fig. 5.1 Programme PROLOG avec relations familiales

Ici A1,...,Am,A,B sont des littéraux. Les littéraux sont, comme dans PL1, construits à partir de symboles de prédicat avec des termes comme arguments. Comme nous pouvons le voir dans le tableau ci-dessus, dans PROLOG il n'y a pas de négations au sens strict de la logique car le signe d'un littéral est déterminé par sa position dans la clause.

5.1 Systèmes PROLOG et implémentations

Un aperçu des systèmes PROLOG actuels est disponible dans la collection de liens sur la page d'accueil de ce livre. Au lecteur, nous recommandons les systèmes très puissants et librement disponibles (sous licences publiques GNU) GNU-PROLOG [Dia04] et SWI-PROLOG. Pour les exemples suivants, SWI-PROLOG [Wie04] a été utilisé.

La plupart des systèmes PROLOG modernes fonctionnent avec un interpréteur basé sur la machine abstraite de Warren (WAM). Le code source PROLOG est compilé en code dit WAM, qui est ensuite interprété par le WAM. Les implémentations les plus rapides d'un WAM gèrent jusqu'à 10 millions d'inférences logiques par seconde (LIPS) sur un PC 1 GHz.

5.2 Exemples simples

Nous commençons avec les relations familiales de l'exemple 3.2 à la page 35. La petite base de connaissances KB est codée—sans les faits pour le prédicat femelle—comme un programme PROLOG nommé rel.pl dans la figure 5.1.

Le programme peut être chargé et compilé dans l'interpréteur PROLOG avec la commande

```
?- [rel].
```

Une requête initiale renvoie la boîte de dialogue

?- enfant (veille, oscar, anne).

Oui

avec la bonne réponse Oui. Comment cette réponse vient-elle ? Pour la requête "?- enfant(eve,oscar,anne)." il y a six faits et une règle avec le même prédicat dans son en-tête de clause. Maintenant, l'unification est tentée entre la requête et chacun des littéraux complémentaires dans les données d'entrée dans l'ordre d'occurrence. Si l'une des alternatives échoue, cela entraîne un retour en arrière jusqu'au dernier point de branchement, et l'alternative suivante est testée. Parce que l'unification échoue avec chaque fait, la requête est unifiée avec la règle récursive de la ligne 8. Maintenant, le système tente de résoudre le sous-objectif enfant(eve,anne,oscar), qui réussit avec la troisième alternative.
La requête

?- descendant(X,Y).

X = oscar
Y = Karen

Oui

est répondu avec la première solution trouvée, comme c'est le cas

?- descendant(clyde,Y).

Y = marie

Oui

La requête

?- descendant(clyde,karen).

n'est cependant pas répondu. La raison en est la clause de la ligne 8, qui spécifie la symétrie du prédicat enfant. Cette clause s'appelle elle-même récursivement sans possibilité de terminaison. Ce problème peut être résolu avec le nouveau programme suivant (les faits ont été omis ici).

```
1 descendant(X,Y) :- enfant(X,Y,Z). 2 descendant(X,Y) :-
enfant(X,Z,Y). 3 descendant(X,Y) :- enfant(X,U,V),
descendant(U,Y).
```

Mais maintenant la requête

?- enfant (veille, oscar, anne).

n'est plus correctement répondu car la symétrie de l'enfant dans les deux dernières variables n'est plus donnée. Une solution aux deux problèmes se trouve dans le programme

```
1 enfant_fait (oscar, karen, franz). 2 child_fact (marie,
karen, franz). 3 fait_enfant(eva,anne,oscar). 4
child_fact(henry,anne,oscar). 5
child_fact(isolde,anne,oscar). 6
child_fact(clyde,marie,oscarb). 7 8 enfant(X,Z,Y) :-
enfant_fait(X,Y,Z). 9 enfant(X,Z,Y) :- child_fact(X,Z,Y).

10 11 descendant(X,Y) :- enfant(X,Y,Z). 12 descendant(X,Y) :-
enfant(X,U,V), descendant(U,Y).
```

En introduisant le nouveau prédicat child_fact pour les faits, le prédicat child n'est plus récursif. Cependant, le programme n'est plus aussi élégant et simple que la première variante — logiquement correcte — de la Fig. 5.1 à la page 68, qui conduit à la boucle infinie. Le programmeur PROLOG doit, comme dans les autres langages, faire attention au traitement et éviter les boucles infinies. PROLOG est juste un langage de programmation et non un démonstrateur de théorèmes.

Il faut ici distinguer sémantique déclarative et procédurale des programmes PROLOG. La sémantique déclarative est donnée par l'interprétation logique des clauses de corne. La sémantique procédurale, au contraire, est définie par l'exécution du programme PROLOG, que nous souhaitons observer plus en détail maintenant. L'exécution du programme de la Fig. 5.1 à la page 68 avec la requête child(eve,oscar,anne) est représentée à la Fig. 5.2 à la page 71 sous la forme d'un arbre de recherche.1 L'exécution commence en haut à gauche avec la requête. Chaque arête représente une étape de résolution SLD possible avec un littéral unifiable complémentaire. Alors que l'arbre de recherche devient infiniment profond par la règle récursive, l'exécution de PROLOG se termine parce que les faits se produisent avant la règle dans les données d'entrée.

Avec la requête descendant(clyde,karen), au contraire, l'exécution de PROLOG ne se termine pas. Nous pouvons le voir clairement dans l' arbre et-ou présenté à la Fig. 5.3 à la page 71. Dans cette représentation, les branches, représentées par mènent de la , tête d'une clause aux sous-buts. Étant donné que tous les sous-objectifs d'une clause doivent être résolus, ce sont des branches et . Toutes les autres branches sont des branches ou, dont au moins une doit être unifiable avec ses nœuds parents. Les deux faits esquissés représentent

1Les constantes ont été abrégées pour gagner de la place.

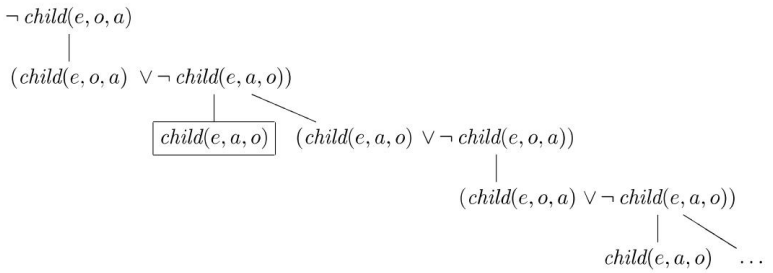


Fig. 5.2 Arbre de recherche PROLOG pour enfant(eve,oscar,anne)

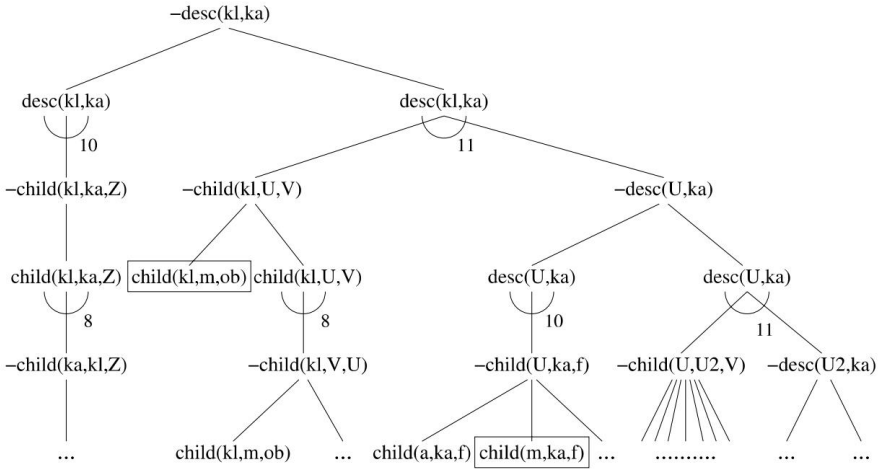


Fig. 5.3 Arbre Et-ou pour desc(clyde,karen)

solution à la requête. L'interpréteur PROLOG ne se termine cependant pas ici, car il fonctionne en utilisant une recherche en profondeur d'abord avec retour en arrière (voir la section 6.2.2) et choisit donc d'abord le chemin infiniment profond vers l'extrême gauche.

5.3 Contrôle de l'exécution et éléments procéduraux

Comme nous l'avons vu dans l'exemple de la relation familiale, il est important de contrôler l'exécution de PROLOG. Éviter les retours en arrière inutiles, en particulier, peut entraîner de fortes augmentations de l'efficacité. L'un des moyens à cette fin est la coupe. En insérant un point d'exclamation dans une clause, nous pouvons éviter de revenir en arrière sur ce point. Dans le programme suivant, le prédicat max(X,Y,Max) calcule le maximum des deux nombres X et Y.

```
1 max(X,Y,X) :- X >= Y. 2 max(X,Y,Y) :-
X < Y.
```

Si le premier cas (première clause) s'applique, alors le second ne sera pas atteint. En revanche, si le premier cas ne s'applique pas, alors la condition du second cas est vraie, ce qui signifie qu'elle n'a pas besoin d'être vérifiée. Par exemple, dans la requête

```
?- max(3,2,Z), Z > 10.
```

le retour arrière est utilisé car Z=3, et la deuxième clause est testée pour max, ce qui est voué à l'échec. Il n'est donc pas nécessaire de revenir en arrière sur cet endroit. Nous pouvons optimiser cela avec une coupe :

```
1 max(X,Y,X) :- X >= Y, !. 2 max(X,Y,Y).
```

Ainsi, la deuxième clause n'est appelée que si elle est vraiment nécessaire, c'est-à-dire si la première clause échoue. Cependant, cette optimisation rend le programme plus difficile à comprendre.

Une autre possibilité de contrôle d'exécution est l'échec du prédicat intégré, qui n'est jamais vrai. Dans l'exemple de relation familiale, nous pouvons tout simplement imprimer tous les enfants et leurs parents avec la requête

```
?- child\_fact(X,Y,Z), write(X), write(' est un enfant de '), write(Y), write(' et '), write(Z), write('.' ),
nl, échec.
```

La sortie correspondante est

```
oscar est un enfant de karen et frank. Mary est une enfant de
Karen et Frank. Eve est une fille d'Anne et d'Oscar.

...
Non.
```

où le prédicat nl provoque un saut de ligne dans la sortie. Quelle serait la sortie à la fin sans l'utilisation du prédicat fail ?

Avec la même base de connaissances, la requête "?- child_fact(ulla,X,Y)." entraînerait la réponse Non car il n'y a pas de faits sur ulla. Cette réponse n'est pas logiquement correcte. Plus précisément, il n'est pas possible de prouver qu'il n'y a pas d'objet avec le nom ulla. Ici, le démonstrateur E répondrait correctement "Aucune preuve trouvée". Ainsi si PROLOG répond Non, cela signifie seulement que la requête Q ne peut pas

être prouvé. Pour cela, cependant, ¬Q ne doit pas nécessairement être prouvé. Ce comportement est appelé négation comme échec.

Se limiter aux clauses de Horn ne pose pas de gros problèmes dans la plupart des cas. Cependant, il est important pour l'exécution procédurale utilisant la résolution SLD (Section 2.5). Grâce au littéral positif déterminé individuellement par clause, à la résolution SLD et, par conséquent, à l'exécution des programmes PROLOG, vous disposez d'un point d'entrée unique dans la clause. C'est le seul moyen d'avoir une exécution reproductible des programmes logiques et, par conséquent, une sémantique procédurale bien définie.

En effet, il existe certainement des énoncés de problème qui ne peuvent pas être décrits par des clauses de Horn. Un exemple est le paradoxe de Russell de l'exemple 3.7 à la page 45, qui contient la clause non-Horn (shaves(barber,X) shaves(X, X)).

5.4 Listes

En tant que langage de haut niveau, PROLOG possède, comme le langage LISP, le type de données liste générique pratique. Une liste avec les éléments A, 2, 2, B, 3, 4, 5 a la forme

```
[A,2,2,B,3,4,5]
```

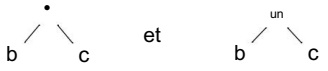
La construction [Head|Tail] sépare le premier élément (Head) du reste (Tail) de la liste. Avec la base de connaissances

```
liste([A,2,2,B,3,4,5]).
```

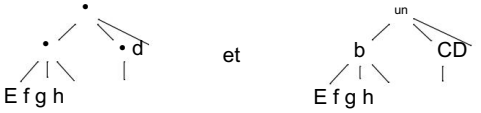
PROLOG affiche la boîte de dialogue

```
?- liste([H|T]).

H=A
T = [2, 2, B, 3, 4, 5]

Oui
```

En utilisant des listes imbriquées, nous pouvons créer des arborescences arbitraires. Par exemple, les deux arbres



peut être représenté par les listes [b,c] et [a,b,c], respectivement, et les deux arbres



par les listes [[e,f,g],[h],d] et [a,[b,e,f,g],[c,h],d], respectivement.
Dans les arbres où les nœuds internes contiennent des symboles, le symbole est la tête du list et les nœuds enfants sont la queue.

Un bel exemple élégant de traitement de liste est la définition du prédicat append(X,Y,Z) pour ajouter la liste Y à la liste X. Le résultat est enregistré dans Z. Le le programme PROLOG correspondant lit

```
1 ajouter([],L,L).
2 ajouter([X|L1],L2,[X|L3]) :- ajouter(L1,L2,L3).
```

Il s'agit d'une description logique déclarative (récursive) du fait que L3 résulte de ajouter L2 à L1. En même temps, cependant, ce programme fait également le travail quand il est appelé. L'appel

```
?- ajouter([a,b,c],[d,1,2],Z).
```

renvoie la substitution Z = [a, b, c, d, 1, 2], tout comme l'appel

```
?- ajouter(X,[1,2,3],[4,5,6,1,2,3]).
```

donne la substitution X = [4, 5, 6]. Ici, nous observons que append n'est pas un fonction à deux places, mais une relation à trois places. En fait, nous pouvons également entrer le "paramètre de sortie" Z et demandez s'il peut être créé.

L'inversion de l'ordre des éléments d'une liste peut également être décrite avec élégance et si programmés simultanément par le prédicat récursif

```
1 nrev([],[]).
2 nrev([H|T],R) :- nrev(T,RT), append(RT,[H],R).
```

qui réduit l'inversion d'une liste à l'inversion d'une liste qui est un élément plus petit. En effet, ce prédicat est très inefficace en raison de l'appel à append. Ce programme est connu sous le nom de naive reverse et est souvent utilisé comme benchmark PROLOG (voir Exercice 5.6 à la page 81). Les choses vont mieux quand on procède avec un temporaire magasin, appelé accumulateur, comme suit :

Liste	Accumulateur
[a B c d] []	
[b,c,d]	[un]
[CD]	[b, une]
[d]	[c,b,a]
[]	[d,c,b,a]

Le programme PROLOG correspondant lit

```
1 accrev([],A,A). 2 accrev([H|
T],A,R) :- accrev(T,[H|A],R).
```

5.5 Programmes auto-modifiables

Les programmes PROLOG ne sont pas entièrement compilés, mais plutôt interprétés par le WAM. Il est donc possible de modifier les programmes en cours d'exécution. Un programme peut même se modifier. Avec des commandes telles que assert et retract, des faits et des règles peuvent être ajoutés à la base de connaissances ou en être retirés.

Une application simple de l'asserta variant est l'ajout de faits dérivés au début de la base de connaissances dans le but d'éviter une dérivation répétée, potentiellement coûteuse en temps (voir l'exercice 5.8 à la page 81) . Si dans notre exemple de relation familiale, nous remplaçons les deux règles pour le descendant du prédicat par

```
1 :- descendant dynamique/2. 2
descendant(X,Y) :- enfant(X,Y,Z), asserta(descendant(X,Y)). 3 descendant(X,Y) :- enfant(X,U,V),
descendant(U,Y), 4 asserta(descendant(X,Y)).
```

alors tous les faits dérivés pour ce prédicat sont enregistrés dans la base de connaissances et ne sont donc pas redérivés à l'avenir. La requête

```
?- descendant(clyde, karen).
```

conduit à l'addition des deux faits

```
descendant(clyde, karen).
descendant(marie, karen).
```

En manipulant des règles avec assert et retract, même des programmes qui se modifient complètement peuvent être écrits. Cette idée est devenue connue sous le terme de programmation génétique. Il permet la construction de programmes d'apprentissage arbitrairement flexibles. En pratique, cependant, il s'avère qu'en raison du grand nombre de modifications possibles insensées, la modification du code par essais et erreurs conduit rarement à une augmentation des performances. Le changement systématique des règles, d'autre part, rend la programmation tellement plus complexe que, jusqu'à présent, de tels programmes qui modifient en profondeur leur propre code n'ont pas réussi. Au Chap. 8, nous montrerons comment l'apprentissage automatique a été assez réussi. Cependant, seules des modifications très limitées du code du programme sont effectuées ici.

```
1 début :- action(état(gauche,gauche,gauche,gauche),
2                 état (droite, droite, droite, droite)).
3
4 actions (début, objectif): -
                plan(Début,Objectif,[Début],Chemin),
5 6 nl,write('Solution :'),nl,
7 write_path(Chemin).
8 % write_path(Path), échec. 9                    % de sortie de toutes les solutions

10 plans (début, objectif, visité, chemin): -
11 aller (Démarrer, Suivant),
12 coffre-fort (Suivant),
13 \+ membre(Suivant,Visité), % non(membre(...))
14              plan(Suivant,Objectif,[Suivant|Visité],Chemin).
15 plans (but, but, chemin, chemin).
16
17 go(state(X,X,Z,K),state(Y,Y,Z,K)):-across(X,Y). % fermier, loup
18 go(state(X,W,X,K),state(Y,W,Y,K)):-across(X,Y). % agriculteur, chèvre
19 go(state(X,W,Z,X),state(Y,W,Z,Y)):-across(X,Y). % fermier, chou
20 go(state(X,W,Z,K),state(Y,W,Z,K)):-across(X,Y). % fermier
21
22 à travers (gauche, droite).
23 à travers (droite, gauche).
24
25 sûr(état(B,W,Z,K)):- à travers(W,Z), à travers(Z,K).
26 sûr(état(B,B,B,K)).
27 sûr(état(B,W,B,B)).
```

Fig. 5.4 Programme PROLOG pour le problème fermier–loup–chèvre–chou

5.6 Un exemple de planification

Exemple 5.1 L'énigme suivante sert d'énoncé de problème pour un programme PRO LOG typique.

> Un fermier veut amener un chou, une chèvre et un loup de l'autre côté d'une rivière, mais son bateau est si petit qu'il ne peut les faire traverser qu'un à la fois. Le fermier réfléchit puis dit à lui-même : « Si j'amène d'abord le loup de l'autre côté, la chèvre mangera le chou. Si je transporter le chou d'abord, puis la chèvre sera mangée par le loup. Que dois-je faire?"

Il s'agit d'une tâche de planification que nous pouvons rapidement résoudre avec un peu de réflexion. Le Le programme PROLOG donné à la Fig. 5.4 n'est pas créé aussi rapidement.

Le programme fonctionne sur les termes de l'état de forme (fermier, loup, chèvre, Chou), qui décrivent l'état actuel du monde. Les quatre variables avec les valeurs possibles gauche, droite donnent l'emplacement des objets. Le plan de prédicat récursif central crée d'abord un état successeur Ensuite, en utilisant go, teste sa sécurité avec safe, et le répète de manière récursive jusqu'à ce que les états de départ et d'objectif soient identiques (dans la ligne de programme 15). Les états qui ont déjà été visités sont stockés dans le troisième argument du plan. Avec le membre de prédicat intégré, il est testé si le state Next a déjà été visité. Si oui, il n'est pas tenté. La définition de la le prédicat write_path pour la tâche de sortie du plan trouvé manque ici. Il est proposé comme exercice pour le lecteur (Exercice 5.2 à la page 80). Pour la pro-

gram teste le littéral write_path(Path) peut être remplacé par write(Path).
Pour la requête "?- start." nous obtenons la réponse

Solution:
Fermier et chèvre de gauche à droite
Fermier de droite à gauche
Fermier et loup de gauche à droite
Fermier et chèvre de droite à gauche
Fermier et chou de gauche à droite
Fermier de droite à gauche
Fermier et chèvre de gauche à droite

Oui

Pour une meilleure compréhension nous décrivons la définition de plan en logique :

z plan(z, z) s z n [go(s, n) safe(n) plan(n, z) plan(s, z)]

Cette définition est nettement plus concise que dans PROLOG. Il y a deux raisons à cela. D'une part, la sortie du plan découvert est sans importance pour la logique. De plus, il n'est pas vraiment nécessaire de vérifier si l'état suivant a déjà été visité si des déplacements inutiles ne gênent pas l'agriculteur. Si, cependant, \+ membre(...) est omis du programme PROLOG, alors il y a une boucle infinie et PROLOG peut ne pas trouver de planning même s'il y en a un. La cause en est la stratégie de recherche de chaînage vers l'arrière de PROLOG, qui, selon le principe de la recherche en profondeur d'abord (Sect. 6.2.2), fonctionne toujours sur les sous-buts un à la fois sans restreindre la profondeur de récursivité, et est donc incomplète. Cela n'arriverait pas à un démonstrateur de théorème avec un calcul complet.

Comme dans toutes les tâches de planification, l'état du monde change au fur et à mesure que les actions sont exécutées d'une étape à l'autre. Cela suggère d'envoyer l'état en tant que variable à tous les prédicats qui dépendent de l'état du monde, comme dans le prédicat safe. Les transitions d'état se produisent dans le prédicat go. Cette approche est appelée calcul de situation [RN10]. Nous nous familiariserons avec une extension intéressante de l'apprentissage des séquences d'action dans des mondes partiellement observables et non déterministes au Chap. dix.

5.7 Programmation en logique de contrainte

La programmation de systèmes d'ordonnancement, dans lesquels de nombreuses conditions logiques et numériques (parfois complexes) doivent être remplies, peut être très coûteuse et difficile avec les langages de programmation conventionnels. C'est précisément là que la logique pourrait être utile. On écrit simplement toutes les conditions logiques dans PL1 puis on entre une requête. Habituellement, cette approche échoue lamentablement. La raison en est le problème du pingouin discuté dans la Sect. 4.3. Le fait que le pingouin(tweety) garantit que le pingouin(tweety) es

vrai. Cependant, cela n'exclut pas que le corbeau (tweety) soit également vrai. Exclure cela avec des axiomes supplémentaires est très gênant (Section 4.3).

La programmation logique par contraintes (CLP), qui permet la formulation explicite de contraintes pour des variables, offre un mécanisme élégant et très efficace pour résoudre ce problème. L'interprète surveille en permanence l'exécution du programme pour le respect de toutes ses contraintes. Le programmeur est entièrement déchargé de la tâche de contrôler les contraintes, ce qui, dans de nombreux cas, peut grandement simplifier la programmation.
Ceci est exprimé dans la citation suivante d'Eugene C. Freuder de [Fre97] :

> La programmation par contraintes représente l'une des approches les plus proches que l'informatique ait jamais faites du Saint Graal de la programmation : l'utilisateur énonce le problème, l'ordinateur le résout.

Sans entrer dans la théorie du problème de satisfaction de contraintes (CSP), nous allons appliquer le mécanisme CLP de GNU-PROLOG à l'exemple suivant.

Exemple 5.2 Le secrétaire de l'école secondaire Albert Einstein doit proposer un plan d'attribution des salles pour les examens finaux. Il a les informations suivantes : les quatre professeurs Mayer, Hoover, Miller et Smith donnent des tests pour les matières Allemand, Anglais, Mathématiques et Physique dans les salles numérotées par ordre croissant 1, 2, 3 et 4. Chaque professeur donne un test pour exactement un sujet dans exactement une pièce. En plus de cela, il connaît les choses suivantes sur les enseignants et leurs matières.

1. M. Mayer ne teste jamais dans la salle 4.
2. M. Miller teste toujours l'allemand.
3. M. Smith et M. Miller ne donnent pas de tests dans les pièces voisines.
4. Mme Hoover teste les mathématiques.
5. La physique est toujours testée dans la salle numéro 4.
6. L'allemand et l'anglais ne sont pas testés en salle 1.

Qui donne un test dans quelle salle ?

Un programme GNU-PROLOG pour résoudre ce problème est donné dans la Fig. 5.5 à la page 79. Ce programme fonctionne avec les variables Mayer, Hoover, Miller, Smith ainsi que German, English, Math, Physics, qui peuvent chacune prendre un nombre entier valeur de 1 à 4 comme numéro de chambre (lignes de programme 2 et 5). Un lien Mayer = 1 et Allemand = 1 signifie que M. Mayer donne le test d'allemand en salle 1. Les lignes 3 et 6 garantissent que les quatre variables particulières prennent des valeurs différentes. La ligne 8 garantit que toutes les variables reçoivent une valeur concrète dans le cas d'une solution. Cette ligne n'est pas absolument nécessaire ici. S'il y avait plusieurs solutions, cependant, seuls les intervalles seraient produits. Aux lignes 10 à 16, les contraintes sont données et les lignes restantes produisent les numéros de salle pour tous les enseignants et toutes les matières dans un format simple.

Le programme est chargé dans GNU-PROLOG avec "['raumplan.pl'].", et
avec "démarrer". on obtient la sortie

[3,1,2,4] [2,3,1,4]

```
1 départ :-
2 fd_domain([Mayer, Hoover, Miller, Smith],1,4),
3 fd_all_different([Mayer, Miller, Hoover, Smith]),
4
5 fd_domain([Allemand, Anglais, Mathématiques, Physique],1,4),
6 fd_all_different([Allemand, Anglais, Mathématiques, Physique]),
7
8 fd_labeling([Mayer, Hoover, Miller, Smith]),
9
10 Mayer #\=4, 11 Miller #=                          % Mayer pas dans la chambre 4
Allemand, 12 dist(Miller,Smith) #>= 2, 13           % Miller teste l'allemand
Hoover #= Mathématiques, 14 Physique #= 4, 15 Allemand   % Distance Miller/Smith >= 2
#\=1, 16 Anglais #\=1 , 17 NL,                      % Hoover teste les mathématiques
                                                    % Physique en salle 4
                                                    % Allemand pas en chambre 1
                                                    % anglais pas en chambre 1

18 écrire([Mayer, Hoover, Miller, Smith]), nl,
19 write([Allemand, Anglais, Mathématiques, Physique]), nl.
```

Fig. 5.5 Programme CLP pour le problème de programmation de salle

Représenté un peu plus commodément, nous avons le programme de salle suivant :

Numéro de chambre	1	2	3	4
Professeur	Aspirateur	Miller	Mayer-Smith	
Sujet	Mathématiques	Allemand	Anglais	Physique

GNU-PROLOG a, comme la plupart des autres langages CLP, un domaine dit fini Solveur de contraintes, avec lequel les variables peuvent être affectées à une plage finie d'entiers. Cela ne doit pas nécessairement être un intervalle comme dans l'exemple. Nous pouvons également entrer une liste de valeurs. A titre d'exercice, l'utilisateur est invité, dans l'exercice 5.9 à la page 81, à créer un Programme CLP, par exemple avec GNU-PROLOG, pour un puzzle logique pas si simple. Ce puzzle, soi-disant créé par Einstein, peut très facilement être résolu avec un CLP système. Si on essayait d'utiliser PROLOG sans contraintes, en revanche, on pourrait facilement grincer des dents. Quiconque trouve une solution élégante avec PROLOG ou un prouveur, s'il vous plaît laissez-le trouver son chemin jusqu'à l'auteur.

5.8 Résumé

Unification, listes, programmation déclarative et vue relationnelle des procédures, dans lequel un argument d'un prédicat peut servir à la fois d'entrée et de sortie, permettent le développement de programmes courts et élégants pour de nombreux problèmes. De nombreux programmes seraient nettement plus long et donc plus difficile à comprendre s'il est écrit dans un langage procédural langue. De plus, ces fonctionnalités du langage font gagner du temps au programmeur. PROLOG est donc aussi un outil intéressant pour le prototypage rapide, notamment pour l'IA

applications. L'extension CLP de PROLOG est utile non seulement pour les puzzles logiques, mais également pour de nombreuses tâches d'optimisation et de planification.

Depuis son invention en 1972, PROLOG est devenu en Europe l'un des principaux langages de programmation en IA en Europe, avec les langages procéduraux. Aux États-Unis, en revanche, le langage inventé nativement LISP domine le marché de l'IA.

PROLOG n'est pas un démonstrateur de théorèmes. Ceci est intentionnel, car un programmeur doit être capable de contrôler facilement et de manière flexible le traitement, et n'irait pas très loin avec un démonstrateur de théorèmes. D'un autre côté, PROLOG n'est pas très utile en soi pour prouver des théorèmes mathématiques. Cependant, il existe certainement des démonstrateurs de théorèmes intéressants qui sont programmés en PROLOG.

La littérature avancée recommandée est [Bra11] et [CM94], ainsi que la manuels [Wie04, Dia04] et, sur le thème du CLP, [Bar98].

5.9 Exercices

Exercice 5.1 Essayez de prouver le théorème de la Sect. 3.7 sur l'égalité des éléments neutres à gauche et à droite des semi-groupes avec PROLOG. Quels problèmes surgissent ?
Quelle est la cause de cela?

Exercice 5.2 (a)

Écrivez un prédicat write_move(+State1, +State2), qui produit une phrase comme « Fermier et loup se croisent de gauche à droite » pour chaque traversée de bateau. State1 et State2 sont des termes de la forme state(Farmer, Wolf, Goat, Cabbage). (b) Écrivez un prédicat récursif write_path(+Path), qui appelle le prédicat
write_move(+State1, +State2) et affiche toutes les actions du fermier.

Exercice 5.3 (a)

À première vue, la variable Path dans le plan des prédicats du programme PROLOG de l'Exemple 5.1 à la page 76 est inutile car elle n'est apparemment modifiée nulle part. A quoi sert-il ?

(b) Si nous ajoutons un échec à la fin de l'action dans l'exemple, toutes les solutions seront données en sortie. Pourquoi chaque solution est-elle maintenant imprimée deux fois ? Comment pouvez-vous empêcher cela?

Exercice 5.4 (a)

Montrer en testant que le démonstrateur de théorème E (contrairement à PROLOG), étant donné la base de connaissances de la Fig. 5.1 page 68, répond à la requête « ?- descendant(clyde, karen) ». correctement. Pourquoi donc? (b) Comparez les réponses de
PROLOG et E pour la requête « ?- descendant(X,
Y).».

Exercice 5.5 Écrivez un programme PROLOG aussi court que possible qui produit 1024 unités.

Exercice 5.6 Étudiez le comportement à l'exécution du prédicat inverse naïf.
(a) Exécutez PROLOG avec l'option trace et observez les appels récursifs de nrev, append et accrev.

(b) Calculez la complexité temporelle asymptotique de append(L1,L2,L3), c'est-à-dire la dépendance du temps d'exécution sur la longueur de la liste pour les grandes listes. Supposons que l'accès à la tête d'une liste arbitraire prend un temps constant. (c) Calculer la complexité temporelle de nrev(L,R). (d) Calculer la complexité temporelle de accrev(L,R). (e) Déterminer expérimentalement la complexité temporelle des prédicats nrev, append et accrev, par exemple en effectuant des mesures de temps (time(+Goal) donne les inférences et le temps CPU.).

Exercice 5.7 Utilisez des symboles de fonction au lieu de listes pour représenter les arbres donnés dans la Sect. 5.4 à la page 73.

Exercice 5.8 La suite de Fibonacci est définie récursivement par fib(0) = 1, fib(1) = 1 et fib(n) = fib(n − 1) + fib(n − 2). (a) Définir un prédicat PROLOG récursif fib(N,R) qui calcule fib(N) et le renvoie dans R.

(b) Déterminer la complexité d'exécution du prédicat fib théoriquement et par la mesure.

(c) Modifiez votre programme en utilisant asserta de manière à ce que les inférences inutiles ne soient plus effectuées. (d) Déterminer la complexité d'exécution du prédicat modifié théoriquement et par mesure (notez que cela dépend si fib a été précédemment appelé).

(e) Pourquoi fib avec asserta est-il aussi plus rapide lorsqu'il est démarré pour la première fois correctement après le démarrage de PROLOG ?

Exercice 5.9 L'énigme logique typique suivante aurait été écrite par Albert Einstein. De plus, il aurait prétendu que seulement 2 % de la population mondiale est capable de le résoudre. Les déclarations suivantes sont données. • Il y a cinq maisons, chacune peinte d'une couleur différente. • Chaque maison est occupée par une personne de nationalité différente. • Chaque résident préfère une boisson spécifique, fume une marque de cigarette spécifique et possède un animal de compagnie spécifique. • Aucune des cinq personnes ne boit la même chose, ne fume la même chose ou n'a le même animal de compagnie. • Indices : – Le Britannique vit dans la maison rouge.

- Le Suédois a un chien.
- Le Danois aime boire du thé.

– La maison verte est à gauche de la maison blanche.

– Le propriétaire de la serre boit du café.
– La personne qui fume du Pall Mall a un oiseau.
– L'homme qui habite la maison du milieu boit du lait.
– Le propriétaire de la maison jaune fume du Dunhill.
- Le Norvégien vit dans la première maison.
– Le fumeur de Marlboro habite à côté de celui qui a un chat.
– L'homme au cheval habite à côté de celui qui fume du Dunhill.
- Le fumeur Winfield aime boire de la bière.
- Le Norvégien habite à côté de la maison bleue.
– L'Allemand fume des Rothmann.
- Le fumeur de Marlboro a un voisin qui boit de l'eau.

Question : A qui appartient le poisson ? (a)

Résolvez d'abord le puzzle manuellement.

(b) Écrivez un programme CLP (par exemple avec GNU-PROLOG) pour résoudre le puzzle.

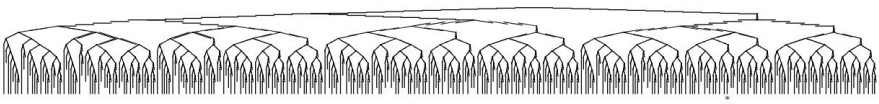
Orientez-vous avec le problème de planification de salle dans la Fig. 5.5 à la page 79.

Recherche, jeux et résolution de problèmes

6

6.1 Introduction

La recherche d'une solution dans un arbre de recherche extrêmement grand pose un problème pour presque tous les systèmes d'inférence. À partir de l'état de départ, il existe de nombreuses possibilités pour la première étape d'inférence. Pour chacune de ces possibilités, il y a encore de nombreuses possibilités dans l'étape suivante, et ainsi de suite. Même dans la preuve d'une formule très simple de [Ert93] avec trois clauses de Horn, chacune avec au plus trois littéraux, l'arbre de recherche pour la résolution SLD a la forme suivante :



L'arbre a été coupé à une profondeur de 14 et a une solution dans le nœud feuille marqué par .
Il n'est possible de le représenter qu'en raison du petit facteur de branchement d'au plus deux et d'une coupure à la profondeur 14. Pour des problèmes réalistes, le facteur de branchement et la profondeur de la première solution peuvent devenir considérablement plus grands.

Supposons que le facteur de branchement est une constante égale à 30 et que la première solution est à une profondeur de 50. L'arbre de recherche a 3050 ≈ 7,2 × 1073 nœuds feuilles. Mais le nombre d'étapes d'inférence est encore plus grand car non seulement chaque nœud feuille, mais aussi chaque nœud interne de l'arbre correspond à une étape d'inférence. Il faut donc additionner les nœuds sur tous les niveaux et obtenir le nombre total de nœuds de l'arbre de recherche

$$30d = \sum_{d=0}^{50} = \frac{1 - 3051}{1 - 30} \approx 7{,}4 \times 1073 ,$$

ce qui ne change pas beaucoup le nombre de nœuds. Évidemment, presque tous les nœuds de cet arbre de recherche sont au dernier niveau. Comme nous le verrons, c'est généralement le cas. Mais revenons maintenant à l'arbre de recherche avec les nœuds 7,4 × 1073 . Supposons que nous disposions de 10 000 ordinateurs capables chacun d'effectuer un milliard d'inférences par seconde, et que nous puissions

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique,
DOI 10.1007/978-0-85729-299-5_6,

distribuer le travail sur tous les ordinateurs sans frais. Le temps de calcul total pour toutes les inférences 7,4 × 1073 serait approximativement égal à

$$7{,}4 \times 1073 \text{ inférences} = \frac{7{,}4 \times 1060 \text{ sec}}{10000 \times 109 \text{ inférences/sec}} \approx 2{,}3 \times 1053 \text{ ans},$$

qui est environ 1043 fois plus de temps que l'âge de notre univers. Par ce simple exercice de réflexion, nous pouvons rapidement reconnaître qu'il n'y a aucune chance réaliste de fouiller complètement ce genre d'espace de recherche avec les moyens dont nous disposons dans ce monde. De plus, les hypothèses liées à la taille de l'espace de recherche étaient tout à fait réalistes. Aux échecs par exemple, il y a plus de 30 coups possibles pour une situation typique, et une partie de 50 demi-tours est relativement courte.

Comment se fait-il alors qu'il y ait de bons joueurs d'échecs - et de nos jours aussi de bons ordinateurs d'échecs ? Comment se fait-il que des mathématiciens trouvent des preuves pour des théorèmes dans lesquels l'espace de recherche est encore plus grand ? De toute évidence, nous, les humains, utilisons des stratégies intelligentes qui réduisent considérablement l'espace de recherche. Le joueur d'échecs expérimenté, tout comme le mathématicien expérimenté, par simple observation de la situation, exclura immédiatement de nombreuses actions comme insensées. Grâce à son expérience, il a la capacité d'évaluer diverses actions pour leur utilité dans l'atteinte de l'objectif. Souvent, une personne passera par la sensation. Si l'on demande à un mathématicien comment il a trouvé une preuve, il peut répondre que l'intuition lui est venue dans un rêve. Dans les cas difficiles, de nombreux médecins trouvent un diagnostic purement au toucher, basé sur tous les symptômes connus. Surtout dans les situations difficiles, il n'y a souvent pas de théorie formelle pour trouver une solution qui garantisse une solution optimale. Dans les problèmes quotidiens, comme la recherche d'un chat en fuite dans la Fig. 6.1 à la page 85, l'intuition joue un grand rôle. Nous traiterons de ce type de méthode de recherche heuristique dans la Sect. 6.3 et décrivent en outre des processus avec lesquels les ordinateurs peuvent, de la même manière que les humains, améliorer leurs stratégies de recherche heuristiques par apprentissage.

Cependant, nous devons d'abord comprendre comment fonctionne la recherche non informée, c'est-à-dire l'essai aveugle de toutes les possibilités. Nous commençons par quelques exemples.

Exemple 6.1 Avec le 8-puzzle, exemple classique d'algorithmes de recherche [Nil98, RN10], les différents algorithmes peuvent être très clairement illustrés. Les carrés numérotés de 1 à 8 sont répartis dans une matrice 3 × 3 comme celle de la Fig. 6.2 à la page 86. Le but est d'atteindre un certain ordre des carrés, par exemple dans l'ordre croissant par rangées comme représenté sur la Fig. 6.2 à la page 86. À chaque étape, un carré peut être déplacé vers la gauche, la droite, le haut ou le bas dans l'espace vide. L'espace vide se déplace donc dans la direction opposée correspondante. Pour l'analyse de l'espace de recherche, il convient de toujours regarder les mouvements possibles du champ vide.

L'arbre de recherche d'un état de départ est représenté sur la Fig. 6.3 page 86. Nous pouvons voir que le facteur de branchement alterne entre deux, trois et quatre. Moyenné sur deux niveaux à la fois, on obtient un facteur de branchement moyen[1] de $\sqrt{8} \approx 2{,}83$. Nous voyons que chaque état est répété plusieurs fois deux niveaux plus profonds car dans une simple recherche non informée, chaque action peut être inversée à l'étape suivante.

[1] Le facteur de ramification moyen d'un arbre est le facteur de ramification qu'aurait un arbre avec un facteur de ramification constant, une profondeur égale et un nombre égal de nœuds feuilles.

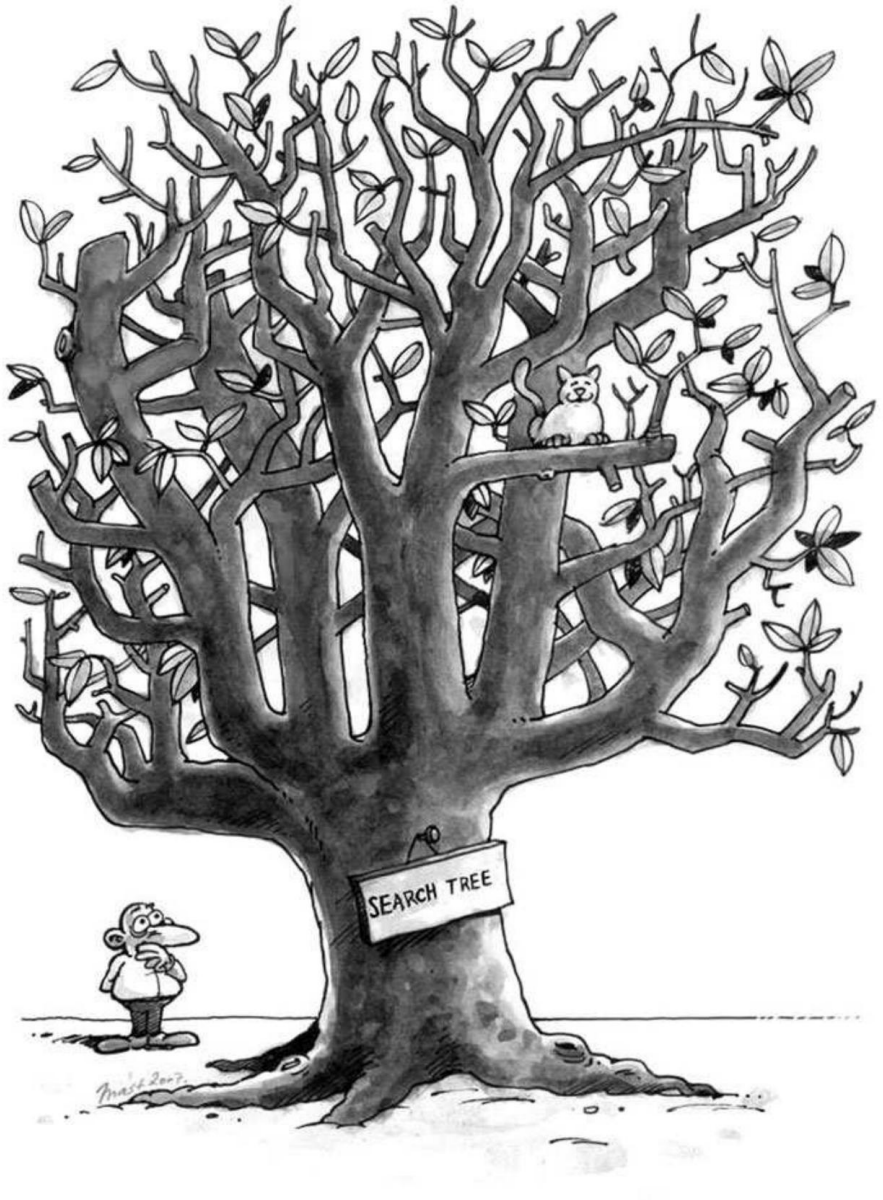


Fig. 6.1 Un arbre de recherche très rogné — ou : « Où est mon chat ? »

Si nous interdisons les cycles de longueur 2, alors pour le même état de départ, nous obtenons le arbre de recherche représenté sur la Fig. 6.4 à la page 86. Le facteur de branchement moyen est réduit d'environ 1 et devient 1.8.2

2Pour un puzzle en 8, le facteur de branchement moyen dépend de l'état de départ (voir l'exercice 6.2 sur pages 110).

Fig. 6.2 États de départ et d'arrivée possibles du 8-puzzle

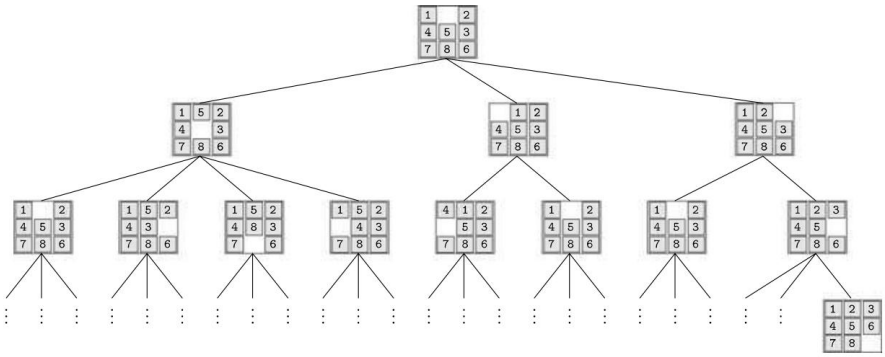


Fig. 6.3 Arbre de recherche pour le 8-puzzle. En bas à droite un état but en profondeur 3 est représenté. Pour économiser de l'espace, les autres nœuds à ce niveau ont été omis

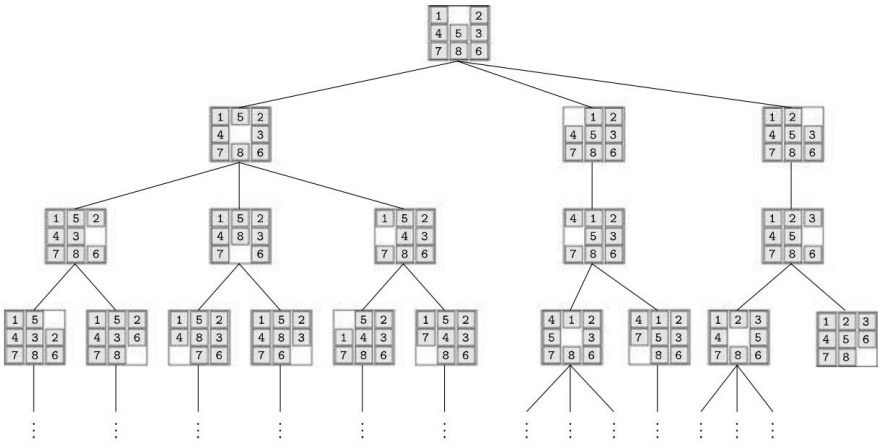


Fig. 6.4 Arbre de recherche d'un 8-puzzle sans cycles de longueur 2

Avant de commencer à décrire les algorithmes de recherche, quelques nouveaux termes sont nécessaires. Nous traitons ici de problèmes de recherche discrète. Étant dans l'état s, une action a1 = a1(s). un nouvel état s = a2(s). ′. Ainsi s ′ Une action différente peut conduire à l'état s ″, in conduit à autres mots: s ″ Application récursive de toutes les actions possibles à tous les états, en commençant par l'état de départ, donne l' arbre de recherche.

Définition 6.1 Un problème de recherche est défini par les valeurs suivantes État :
Description de l'état du monde dans lequel l'agent de recherche trouve lui-même.
État de démarrage : l'état initial dans lequel l'agent de recherche est démarré.
État d'objectif : si l'agent atteint un état d'objectif, il se termine et produit une solution (si vous le souhaitez).
Actions : tous les agents ont autorisé les actions.
Solution : Le chemin dans l'arborescence de recherche de l'état de départ à l'état d'arrivée.
Fonction de coût : attribue une valeur de coût à chaque action. Nécessaire pour trouver une solution optimale en termes de coût.
Espace d'état : ensemble de tous les états.

Appliqué au 8-puzzle, on obtient
État : matrice 3×3 S avec les valeurs 1, 2, 3, 4, 5, 6, 7, 8 (une fois chacune) et une vide carré.
État de départ : un état arbitraire.
État du but : Un état arbitraire, par exemple l'état donné à droite dans la Fig. 6.2 à la page 86.
Actions : Déplacement de la case vide Sij vers la gauche (si j = 1), la droite (si j = 3), le haut (si i = 1), le bas (si i = 3).
Fonction de coût : La fonction constante 1, puisque toutes les actions ont un coût égal.
Espace d'état : L'espace d'état est dégénéré en domaines mutuellement inaccessibles (Exercice 6.4 page 110). Il existe donc des problèmes insolubles à 8 énigmes.
Pour l'analyse des algorithmes de recherche, les termes suivants sont nécessaires :

Définition 6.2 • Le
nombre d'états successeurs d'un état s est appelé facteur de branchement b(s), ou b si le facteur de branchement est constant.
• Le facteur de branchement effectif d'un arbre de profondeur d avec n nœuds au total est défini comme le facteur de branchement qu'aurait un arbre avec un facteur de branchement constant, une profondeur égale et n égal (voir l'exercice 6.3 à la page 110). • Un algorithme de recherche est dit complet s'il trouve une solution pour chaque problème soluble. Si un algorithme de recherche complet se termine sans trouver de solution, alors le problème est insoluble.

Pour une profondeur d et un nombre de nœuds donnés n, le facteur de branchement effectif peut être calculé en résolvant l'équation

$$n = \frac{b^{d+1} - 1}{b-1} \tag{6.1}$$

La prudence est recommandée si une variable quantifiée apparaît en dehors de la portée de son quantifier, comme par exemple x dans

$$xp(x) \quad xq(x).$$

Ici l'une des deux variables doit être renommée, et dans

$$xp(x) \quad yq(y)$$

le quantificateur peut facilement être mis au premier plan, et on obtient en sortie la formule équivalente

$$x \quad yp(x) \quad q(y).$$

Si, toutefois, nous souhaitons amener correctement le quantificateur au début de

$$(\quad xp(x)) \quad yq(y) \tag{3.4}$$

on écrit d'abord la formule sous la forme équivalente

$$\neg(\quad xp(x)) \quad yq(y).$$

Le premier quantificateur universel se transforme maintenant en

$$(\quad x\neg p(x)) \quad yq(y)$$

et maintenant les deux quantificateurs peuvent enfin être tirés vers l'avant pour

$$x \quad y\neg p(x) \quad q(y),$$

qui équivaut à

$$x \quad yp(x) \quad q(y).$$

Nous voyons alors que dans (3.4) nous ne pouvons pas simplement tirer les deux quantificateurs vers l'avant. Au contraire, nous devons d'abord éliminer les implications afin qu'il n'y ait pas de négations sur les quantificateurs. Il tient en général que nous ne pouvons extraire des quantificateurs que si les négations n'existent directement que sur les sous-formules atomiques.

Exemple 3.5 Comme cela est bien connu en analyse, la convergence d'une série $(a_n)_{n \in N}$ vers une limite a est définie par

$$\varepsilon > 0 \quad n_0 \in N \quad n > n_0 |a_n - a| < \varepsilon .$$

Avec la fonction abs(x) pour |x|, a(n) pour a_n, minus(x, y) pour $x - y$ et les prédicats el(x, y) pour $x = y$, gr(x, y) pour x>y, la formule se lit

ε (gr(ε , 0) n0 (el(n0,N) n (gr(n, n0) gr(ε , abs(moins(a(n), a)))))) . (3.5)

Ce n'est clairement pas sous forme normale de prénex. Étant donné que les variables des quantificateurs internes n0 et n n'apparaissent pas à gauche de leurs quantificateurs respectifs, aucune variable ne doit être renommée. Ensuite, nous éliminons les implications et obtenons

ε (¬gr(ε , 0) n0 (¬el(n0,N) n (¬gr(n, n0) gr(ε , abs(moins(a(n), a)))))).

Parce que chaque négation est devant une formule atomique, nous avançons les quantificateurs, éliminons les parenthèses redondantes, et avec

ε n0 n (¬gr(ε , 0) ¬el(n0,N) ¬gr(n, n0) gr(ε , abs(moins(a(n), a))))

elle devient une clause quantifiée sous forme normale conjonctive.

La formule transformée est équivalente à la formule de sortie. Le fait que cela la transformation est toujours possible est garantie par

Théorème 3.2 Toute formule logique de prédicat peut être transformée en une formule équivalente sous forme normale prénexe.

De plus, nous pouvons éliminer tous les quantificateurs existentiels. Cependant, la formule résultant de la soi-disant skolémisation n'est plus équivalente à la sortie pour mula. Sa satisfaisabilité reste cependant inchangée. Dans de nombreux cas, notamment quand on veut montrer l'insatisfaisabilité de KB ¬Q, cela suffit. La formule suivante sous forme normale prénex sera désormais skolémisée :

x1 x2 y1 x3 y2 p(f (x1), x2, y1) q(y1, x3, y2).

Comme la variable y1 dépend apparemment de x1 et x2, chaque occurrence de y1 est remplacée par une fonction de Skolem g(x1, x2). Il est important que g soit un nouveau symbole de fonction qui n'est pas encore apparu dans la formule. On obtient

x1 x2 x3 y2 p(f (x1), x2, g(x1, x2)) q(g(x1, x2), x3, y2)

et remplacer y2 de manière analogue par h(x1, x2, x3), ce qui conduit à

x1 x2 x3 p(f (x1), x2, g(x1, x2)) q(g(x1, x2), x3, h(x1, x2, x3)).

Parce que maintenant toutes les variables sont universellement quantifiées, les quantificateurs universels peuvent être omis, ce qui entraîne

p(f (x1), x2, g(x1, x2)) q(g(x1, x2), x3, h(x1, x2, x3)).

TRANSFORMATION DE FORME NORMALE(Formule) : 1.
Transformation en forme normale prénexe :
Transformation en forme normale conjonctive (Théorème 2.1) :
Élimination des équivalences.
Élimination des implications.
Application répétée de la loi de Morgan et de la loi distributive.
Renommage des variables si nécessaire.
Factoriser les quantificateurs universels.
2. Skolémisation :
Remplacement des variables existentiellement quantifiées par de nouvelles fonctions de Skolem.
Suppression des quantificateurs universels résultants.

Fig. 3.2 Transformation des formules logiques de prédicat en forme normale

Nous pouvons maintenant éliminer le quantificateur existentiel (et donc aussi le quantificateur universel) dans (3.5) à la page 39 en introduisant la fonction de Skolem n0(ε). Le prénexe skolémisé et la forme normale conjonctive de (3.5) à la page 39 se lit donc

¬gr(ε , 0) ¬el(n0(ε),N) ¬gr(n, n0(ε)) gr(ε , abs(moins(a(n), a))).

En supprimant la variable n0, la fonction Skolem peut recevoir le nom n0.

Lors de la skolémisation d'une formule sous forme normale prénexe, tous les quantificateurs existentiels sont éliminés de l'extérieur vers l'intérieur, où une formule de la forme x1 ... xn y est remplacée par x1 ... xn [y/f (x1,...,xn)], pendant laquelle f peut ne pas apparaître dans .
Si un quantificateur existentiel est à l'extérieur, comme dans yp(y), alors y doit être remplacé par une constante (c'est-à-dire par un symbole de fonction de zéro).

La procédure de transformation d'une formule en forme normale conjonctive est résumée dans le pseudocode représenté sur la figure 3.2. La skolémisation a un temps d'exécution polynomial dans le nombre de littéraux. Lors de la transformation en forme normale, le nombre de littéraux sous forme normale peut croître de façon exponentielle, ce qui peut entraîner un temps de calcul exponentiel et une utilisation exponentielle de la mémoire. La raison en est l'application répétée de la loi distributive. Le problème réel, qui résulte d'un grand nombre de clauses, est l'explosion combinatoire de l'espace de recherche pour une preuve de résolution ultérieure. Cependant, il existe un algorithme de transformation optimisé qui ne génère polynomialement que de nombreux littéraux [Ede91].

3.4 Calculs de preuve

Pour raisonner en logique des prédicats, divers calculs de raisonnement naturel tels que le calcul de Gentzen ou le calcul séquentiel ont été développés. Comme leur nom l'indique, ces calculs sont destinés à être appliqués par des humains, puisque les règles d'inférence sont plus

Tableau 3.2 Preuve simple avec modus ponens et élimination des quantificateurs

WB :	1	enfant(eve, anne, oscar)
WB :	2	x y z enfant(x, y, z) enfant(x, z, y)
E(2) : x/eve,y/anne, z/oscar	3	enfant(eve, anne, oscar) enfant(eve, oscar, anne)
MP(1, 3)	4	enfant(eve, oscar, anne)

ou moins intuitif et les calculs fonctionnent sur des formules PL1 arbitraires. Dans la section suivante nous nous concentrerons principalement sur le calcul de résolution, qui est en pratique le calcul efficace et automatisable le plus important pour les formules en conjonctive normale formulaire. Ici, en utilisant l'exemple 3.2 de la page 35 , nous donnerons une toute petite preuve « naturelle ». On utilise la règle d'inférence

$$\frac{A,\ A \quad B}{B} \text{ (modus ponens, MP) et } \frac{x\ A}{A[x/t]} \text{ (-élimination, E).}$$

Le modus ponens est déjà familier de la logique propositionnelle. Lors de l'élimination quantificateurs universels, il faut garder à l'esprit que la variable quantifiée x doit être remplacé par un terme fondamental t, c'est-à-dire un terme qui ne contient aucune variable. La preuve de enfant(eve, oscar, anne) à partir d'une base de connaissances réduite de manière appropriée est présenté dans le tableau 3.2.

Les deux formules de la base de connaissances réduite sont répertoriées dans les lignes 1 et 2. Dans ligne 3, les quantificateurs universels de la ligne 2 sont éliminés, et dans la ligne 4, la revendication est dérivé avec modus ponens.

Le calcul consistant en les deux règles d'inférence données n'est pas complet. Cependant, il peut être étendu à une procédure complète en ajoutant d'autres inférences règles. Ce fait non trivial est d'une importance fondamentale pour les mathématiques et l'IA. Le logicien autrichien Kurt Gödel a prouvé en 1931 que [Göd31a]

> Théorème 3.3 (Théorème de complétude de Gödel) La logique des prédicats du premier ordre est complet. C'est-à-dire qu'il existe un calcul avec lequel toute proposition qui est une conséquence d'une base de connaissances KB peut être prouvée. Si Ko | , alors il tient que KB .

Toute proposition vraie dans la logique des prédicats du premier ordre est donc démontrable. Mais est l'inverse aussi vrai ? Tout ce que nous pouvons déduire syntaxiquement est-il réellement vrai ? Le la réponse est "oui":

> Théorème 3.4 (Correctivité) Il existe des calculs avec lesquels seules des propositions vraies peuvent être prouvées. Autrement dit, si KB est vérifiée, alors KB | .

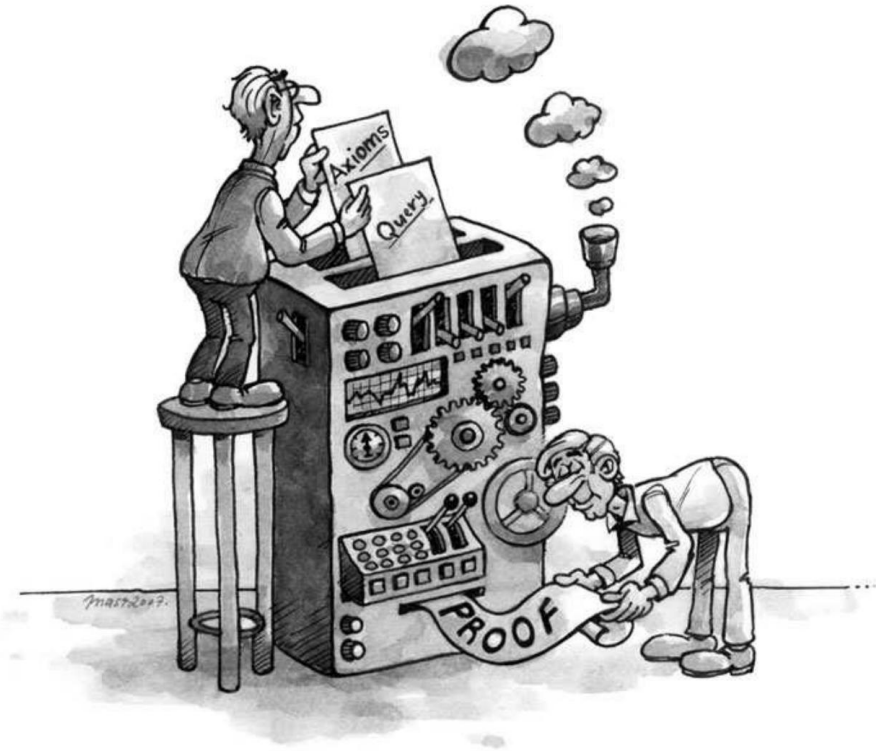


Fig. 3.3 La machine logique universelle

En fait, presque tous les calculs connus sont corrects. Après tout, cela n'a aucun sens de travailler avec des méthodes de preuve incorrectes. La prouvabilité et la conséquence sémantique sont donc concepts équivalents, tant qu'un calcul correct et complet est utilisé. Ainsi la logique des prédicats du premier ordre devient un outil puissant pour les mathématiques et l'IA. Le les calculs de déduction naturelle susmentionnés sont plutôt inadaptés à l'automatisation.

Seul le calcul de résolution, qui a été introduit en 1965 et fonctionne essentiellement avec une seule règle d'inférence simple a permis la construction de puissants démonstrateurs orem automatisés, qui ont ensuite été utilisés comme machines d'inférence pour les systèmes experts.

3.5 Résolution

En effet, le calcul de la résolution correcte et complète a déclenché une euphorie logique au cours des années 1970. De nombreux scientifiques pensaient que l'on pouvait formuler presque toutes les tâches de la représentation des connaissances et du raisonnement en PL1, puis résolvez-le avec un démonstrateur automatisé. La logique des prédicats, un langage puissant et expressif, ainsi qu'un Le calcul de la preuve complète semblait être la machine intelligente universelle pour représenter les connaissances et résoudre de nombreux problèmes difficiles (Fig. 3.3).

Si l'on alimente un ensemble d'axiomes (c'est-à-dire une base de connaissances) et une requête dans une telle machine logique en entrée, la machine recherche une preuve et la renvoie - car il en existe une et elle sera trouvée - en sortie. Avec le théorème de complétude de Gödel et le travail de Herbrand comme fondation, beaucoup a été investi dans la mécanisation de la logique. La vision d'une machine qui pourrait, avec une base de connaissances PL1 arbitraire et non contradictoire, prouver n'importe quelle requête vraie était très séduisante. En conséquence, jusqu'à présent, de nombreux calculs de preuve pour PL1 sont développés et réalisés sous la forme de démonstrateurs de théorèmes. À titre d'exemple, nous décrivons ici le calcul de résolution historiquement important et largement utilisé et montrons ses capacités. La raison du choix de la résolution comme exemple de calcul de preuve dans ce livre est, comme indiqué, son importance historique et didactique. Aujourd'hui, la résolution ne représente qu'un des nombreux calculs utilisés dans les prouveurs hautes performances.

Nous commençons par essayer de compiler la preuve du tableau 3.2 page 41 avec la base de connaissances de l'exemple 3.2 en une preuve de résolution. D'abord les formules sont transformées en forme normale conjonctive et la requête niée

$$\neg Q \equiv \neg enfant(eve, oscar, anne)$$

est ajouté à la base de connaissances, ce qui donne

$$\begin{aligned} KB \quad & \neg Q \equiv (enfant(eve, anne, oscar))1 \\ & (\neg enfant(x, y, z) \quad enfant(x, z, y))2 \\ & (\neg enfant(eve, oscar, anne))3. \end{aligned}$$

La preuve pourrait alors ressembler à quelque chose comme

$$\begin{aligned} (2)\ x/eve, y/anne, z/oscar : & (\neg enfant(eve, anne, oscar) \quad enfant(eve, \\ & oscar, anne))4 \\ & Res(3, 4) : (\neg enfant(eve, anne, oscar))5 \\ & Res(1, 5) : ()6, \end{aligned}$$

où, dans la première étape, les variables x,y,z sont remplacées par des constantes. Ensuite, deux étapes de résolution suivent sous l'application de la règle de résolution générale de (2.2), qui a été reprise telle quelle de la logique propositionnelle.

Les circonstances de l'exemple suivant sont un peu plus complexes. Nous supposons que tout le monde connaît sa propre mère et demandons si Henry connaît quelqu'un.
Avec le symbole de fonction "mère" et le prédicat "sait", nous devons dériver une contradiction de

$$(connaît(x,mère(x)))1 \quad (\neg connaît(henri, y))2.$$

Par le remplacement x/henry, y/mother(henry) on obtient le couple de clauses contradictoires

$$(connaît(henri,mère(henri)))1 \quad (\neg sait(henri,mère(henri)))2.$$

Cette étape de remplacement est appelée unification. Les deux littéraux sont complémentaires, ce qui signifie qu'ils sont les mêmes à l'exception de leurs signes. La clause vide est maintenant dérivable avec une étape de résolution, par laquelle il a été montré que Henry fait connaître quelqu'un (sa mère). Nous définissons

Définition 3.7 Deux littéraux sont dits unifiables s'il existe une substitution σ pour toutes les variables qui rend les littéraux égaux. Un tel σ est appelé unificateur. Un unificateur est appelé l'unificateur le plus général (MGU) si tous les autres unificateurs peuvent être obtenu par substitution de variables.

Exemple 3.6 On veut unifier les littéraux p(f (g(x)), y, z) et p(u, u, f (u)). Plusieurs unificateurs sont

σ1 : y/f (g(x)), z/f (f (g(x))), σ2 : x/h(v), y/f (g(h(v))), z/f (f (g(h(v)))), σ3 : x/h(h(v)), y/f (g(h(h(v)))), z/f (f (g(h(h(v))))), u/f (g(h(v))) u/f (g(h(v)))

σ4 : x/h(a), y/f (g(h(a))), z/f (f (g(h(a)))), σ5 : x/a, y/f (g (a)), z/f (f (g(a))), u/f (g(h(a))) u/f (g(a))

où σ1 est l'unificateur le plus général. Les autres unificateurs résultent de σ1 par le substitutions x/h(v), x/h(h(v)), x/h(a), x/a.

Nous pouvons voir dans cet exemple que lors de l'unification des littéraux, les symboles de prédicat peuvent être traités comme des symboles de fonction. Autrement dit, le littéral est traité comme un terme. Les implémentations des algorithmes d'unification traitent les arguments des fonctions de manière séquentielle. Les termes sont unifiés récursivement sur la structure des termes. Les algorithmes d'unifi cation les plus simples sont très rapides dans la plupart des cas. Dans le pire des cas, cependant, le temps de calcul peut croître de façon exponentielle avec la taille des termes. Parce que pour l'automatisation prouve que le nombre écrasant de tentatives d'unification échoue ou est très simple, en la plupart des cas, la complexité du pire des cas n'a pas d'effet dramatique. L'unification la plus rapide les algorithmes ont une complexité presque linéaire même dans le pire des cas [Bib82].

Nous pouvons maintenant donner la règle générale de résolution pour la logique des prédicats :

Définition 3.8 La règle de résolution pour deux clauses en forme normale conjonctive lit

$$\frac{(A1 \cdots Am \; B), (\neg B' \; C1 \cdots Cn) \; \sigma(B) = \sigma(B')}{(\sigma(A1) \cdots \sigma(Am) \; \sigma(C1) \cdots \sigma(Cn))}, \quad (3.6)$$

où σ est le MGU de B et B′.

Théorème 3.5 La règle de résolution est correcte. Autrement dit, la résolvante est une conséquence sémantique des deux clauses parentes.

Pour l'exhaustivité, cependant, nous avons encore besoin d'un petit ajout, comme indiqué dans le exemple suivant.

Exemple 3.7 Le célèbre paradoxe de Russell se lit comme suit : "Il y a un barbier qui rase tous ceux qui ne se rasent pas." Cette affirmation est contradictoire, c'est-à-dire qu'elle est insatisfaisante. Nous souhaitons le montrer avec résolution. Formalisé en PL1, le paradoxe se lit

x rase(barbier,x) ¬rase(x, x)

et la transformation en forme de clause donne (voir Exercice 3.6 à la page 55)

(¬rase(barbier,x) ¬rase(x, x))1 (rase(barbier,x) rase(x, x))2. (3.7)

De ces deux clauses nous pouvons tirer plusieurs tautologies, mais aucune contradiction. La résolution n'est donc pas complète. Nous avons encore besoin d'une autre règle d'inférence.

Définition 3.9 La factorisation d'une clause est accomplie par

$$\frac{(A1 \quad A2 \quad \cdots \quad An)\ \sigma\ (A1) = \sigma\ (A2)}{(\sigma\ (A2) \quad \cdots \quad \sigma\ (An))},$$

où σ est le MGU de A1 et A2.

Maintenant une contradiction peut être dérivée de (3.7)

Fak(1, σ : x/barbier) : (¬rase(barbier, barbier))3
Fak(2, σ : x/barbier) : (rase(barbier, barbier))4
Rés(3, 4) : ()5

et nous affirmons :

Théorème 3.6 La règle de résolution (3.6) associée à la règle de factorisation (3.9) est réfutation complète. Autrement dit, en appliquant les étapes de factorisation et de résolution, la clause vide peut être dérivée de n'importe quelle formule insatisfaisante sous forme normale conjonctive.

3.5.1 Stratégies de résolution

Alors que l'exhaustivité de la résolution est importante pour l'utilisateur, la recherche d'une preuve peut être très frustrante en pratique. La raison en est l'immense espace de recherche combinatoire. Même s'il n'y a que très peu de paires de clauses dans KB ¬Q au début, le prouveur génère une nouvelle clause à chaque pas de résolution, ce qui augmente le nombre de pas de résolution possibles à l'itération suivante. Ainsi, on a longtemps tenté de réduire l'espace de recherche en utilisant des stratégies spéciales, de préférence sans perdre en complétude. Les stratégies les plus importantes sont les suivantes.

La résolution unitaire donne la priorité aux étapes de résolution dans lesquelles l'une des deux clauses consiste en un seul littéral, appelée clause unitaire. Cette stratégie préserve l'exhaustivité et conduit dans de nombreux cas, mais pas toujours, à une réduction de l'espace de recherche. Il s'agit donc d'un processus heuristique (voir section 6.3).

On obtient une réduction garantie de l'espace de recherche par application de l' ensemble de stratégie de support. Ici, un sous-ensemble de KB ¬Q est défini comme l'ensemble de support (SOS). Chaque étape de résolution doit impliquer une clause du SOS, et le résolvant est ajouté au SOS. Cette stratégie est incomplète. Il devient complet lorsqu'on s'assure que l'ensemble de clauses est satisfiable sans le SOS (voir exercice 3.7 page 55). La requête négative ¬Q est souvent utilisée comme SOS initial.

Dans la résolution d'entrée, une clause de l'ensemble d'entrée KB ¬Q doit être impliquée dans chaque étape de résolution. Cette stratégie réduit également l'espace de recherche, mais au prix de l'exhaustivité.

Avec la règle du littéral pur, toutes les clauses contenant des littéraux pour lesquels il n'y a pas de littéraux complémentaires dans d'autres clauses peuvent être supprimées. Cette règle réduit l'espace de recherche et est complète, elle est donc utilisée par pratiquement tous les prouveurs de résolution.

Si les littéraux d'une clause K1 représentent un sous-ensemble des littéraux de la clause K2, alors K2 peut être supprimé. Par exemple, la clause

(il pleut (aujourd'hui) street_wet (aujourd'hui))

est redondant si street_wet(today) est déjà valide. Cette étape de réduction importante est appelée subsomption. La subsomption, elle aussi, est complète.

3.5.2 Égalité

L'égalité est une cause particulièrement gênante de la croissance explosive de l'espace de recherche. Si nous ajoutons (3.1) à la page 36 et les axiomes d'égalité formulés dans (3.2) à la page 36 à la base de connaissances, alors la clause de symétrie ¬x = y y = x peut être unifiée avec toute équation positive ou négative, par exemple . Cela conduit à la dérivation de nouvelles clauses et équations sur lesquelles les axiomes d'égalité peuvent à nouveau être appliqués, et ainsi de suite. Les axiomes de transitivité et de substitution ont des conséquences similaires. Pour cette raison, des règles d'inférence spéciales pour l'égalité ont été développées qui se passent d'axiomes d'égalité explicites et, en particulier, réduisent l'espace de recherche. Démodulation,

par exemple, permet de substituer un terme t2 à t1, si l'équation t1 = t2 existe. Un
l'équation t1 = t2 est appliquée par unification à un terme t comme suit :

$$\frac{t1 = t2, \qquad (\ldots t \ldots), \sigma(t1) = \sigma(t)}{(\ldots \sigma(t2) \ldots)}.$$

Un peu plus générale est la paramodulation, qui fonctionne avec des équations conditionnelles [Bib82, Lov78].

L'équation t1 = t2 permet la substitution du terme t1 par t2 ainsi que la
substitution t2 par t1. Il est généralement inutile d'annuler une substitution qui a déjà
été réalisée. Au contraire, les équations sont fréquemment utilisées pour simplifier les termes.
Ils sont donc souvent utilisés dans un seul sens. Les équations qui ne sont utilisées que dans
une direction sont appelées équations dirigées. Le traitement efficace des équations dirigées est accompli par
des systèmes dits de réécriture de termes. Pour les formules avec beaucoup
équations il existe des démonstrateurs d'égalité spéciaux.

3.6 Démonstrateurs de théorèmes automatisés

Les implémentations des calculs de preuve sur les ordinateurs sont appelées démonstrateurs de théorèmes. Le long de
avec des prouveurs spécialisés pour des sous-ensembles de PL1 ou des applications particulières, il existe aujourd'hui
toute une gamme de démonstrateurs automatisés pour la logique complète des prédicats et les logiques
d'ordre supérieur, dont seuls quelques-uns seront discutés ici. Un aperçu des plus importants
peuvent être trouvés dans [McC].

L'un des plus anciens prouveurs de résolution a été développé à l'oratoire Argonne National Lab à Chicago.
Basé sur les premiers développements à partir de 1963, Otter [Kal01],
a été créé en 1984. Surtout, Otter a été appliqué avec succès dans des domaines spécialisés
des mathématiques, comme on peut l'apprendre sur sa page d'accueil :

> « Actuellement, la principale application d'Otter est la recherche en algèbre abstraite et en logique formelle.
> Otter et ses prédécesseurs ont été utilisés pour répondre à de nombreuses questions ouvertes dans les domaines de
> semi-groupes finis, algèbre booléenne ternaire, calculs logiques, logique combinatoire, théorie des groupes,
> la théorie des réseaux et la géométrie algébrique.

Plusieurs années plus tard, l'Université de Technologie de Munich a créé le prouveur haute performance
SETHEO [LSBB92] basé sur la technologie rapide PROLOG. Avec
Dans le but d'atteindre des performances encore plus élevées, une implémentation pour ordinateurs parallèles
a été développée sous le nom de PARTHEO. Il s'est avéré qu'il ne valait pas la peine d'utiliser du matériel
spécial dans les démonstrateurs de théorèmes, comme c'est également le cas dans d'autres domaines.
de l'IA, car ces ordinateurs sont très vite dépassés par des processeurs plus rapides et
algorithmes plus intelligents. Munich est également le lieu de naissance de E [Sch02], un prouveur d'équations
moderne primé, avec lequel nous nous familiariserons dans le prochain
exemple. Sur la page d'accueil de E, on peut lire le caractère compact et ironique suivant :

sation, dont la seconde partie s'applique d'ailleurs à tous les démonstrateurs automatisés existants aujourd'hui.

> « E est un démonstrateur de théorème purement équationnel pour la logique clausale. Cela signifie que c'est un programme qui vous pouvez y insérer une spécification mathématique (en logique clausale avec égalité) et une hypothèse, et qui s'exécutera ensuite indéfiniment, utilisant toutes les ressources de votre machine. Très occasionnellement, il trouvera une preuve de l'hypothèse et vous le dira ;-).

Trouver des preuves pour des propositions vraies est apparemment si difficile que la recherche ne réussit que très rarement, ou seulement après un temps très long, voire pas du tout. Nous entrerons dans ceci plus en détail au Chap. 4. Ici, il convient de mentionner, cependant, que non seulement ordinateurs, mais aussi la plupart des gens ont du mal à trouver des preuves formelles strictes.

Bien qu'évidemment les ordinateurs eux-mêmes soient dans de nombreux cas incapables de trouver une preuve, la meilleure chose à faire est de construire des systèmes qui fonctionnent semi-automatiquement. et permettre une coopération étroite avec l'utilisateur. Ainsi l'humain peut mieux appliquer son connaissance de domaines d'application particuliers et peut-être limiter la recherche de la preuve. L'un des démonstrateurs interactifs les plus réussis pour la logique des prédicats d'ordre supérieur est Isabelle [NPW02], un produit commun de l'Université de Cambridge et de l'Université de Technologie, Munich.

Toute personne à la recherche d'un prouveur de haute performance devrait consulter les résultats actuels de la CASC (CADE ATP System Competition) [SS06].1 Ici, nous constatons que le vainqueur de 2001 à 2006 dans les catégories PL1 et clause de forme normale a été Le prouveur de Manchester Vampire, qui fonctionne avec une variante de résolution et un spécial approche de l'égalité. Le système Waldmeister de l'Institut Max Planck à Sarre-bruck est leader depuis des années dans la preuve d'égalité.

Les nombreuses positions de tête des systèmes allemands au CASC montrent que la recherche allemande groupes dans le domaine de la démonstration automatisée de théorèmes jouent un rôle de premier plan, aujourd'hui ainsi que par le passé.

3.7 Exemples mathématiques

Nous souhaitons maintenant démontrer l'application d'un prouveur automatisé avec le prouveur E mentionné ci-dessus [Sch02]. E est un prouveur d'égalité spécialisé qui se rétrécit considérablement l'espace de recherche par un traitement optimisé de l'égalité.

Nous voulons prouver que les éléments neutres à gauche et à droite dans un semi-groupe sont égaux. Nous formalisons d'abord la réclamation étape par étape.

1CADE est la "Conférence annuelle sur la déduction automatisée" [CAD] et ATP signifie "Automated Theorem Prover".

Définition 3.10 Une structure (M,·) constituée d'un ensemble M avec une opération interne à deux places « · » est appelée semi-groupe si la loi d'associativité

$$x \quad y \quad z \; (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

tient. Un élément e M est dit neutre à gauche (neutre à droite) si xe · x = x (xx · e = x).

Il reste à montrer que

Théorème 3.7 Si un semi-groupe a un élément neutre à gauche el et un élément neutre à droite er , alors el = er .

Nous prouvons d'abord le théorème semi-formellement par un raisonnement mathématique intuitif. Il est clair que pour tout x M que

$$el \cdot x = x \tag{3.8}$$

et

$$x \cdot er = x. \tag{3.9}$$

Si on pose x = er dans (3.8) et x = el dans (3.9), on obtient les deux équations el · er = er et el · er = el . La jonction de ces deux équations donne

$$el = el \cdot er = er,$$

que nous voulons prouver. Dans la dernière étape, d'ailleurs, nous avons utilisé le fait que l'égalité est symétrique et transitive.

Avant d'appliquer le prouveur automatisé, nous réalisons manuellement la preuve de résolution. Premièrement, nous formalisons la requête niée et la base de connaissances KB, constituée des axiomes sous forme de clauses en forme normale conjonctive :

(¬el = er)1	requête inversée
(m(m(x, y), z) = m(x, m(y, z)))2 (m(el,x) = x)3 (m(x, er) = x)4	
axiomes d'égalité :	
(x = x)5	(réflexivité)
(¬x = y y = x)6 (¬x = y ¬y = z x = z)7 (¬x = y	(symétrie) (transitivité)
m(x, z) = m (y, z))8 (¬x = y m(z, x) = m(z, y))9	substitution en m substitution en m,

où la multiplication est représentée par le symbole de fonction à deux chiffres m. Les axiomes d'égalité ont été formulés de manière analogue à (3.1) à la page 36 et (3.2) à la page 36.
Une preuve de résolution simple a la forme

Res(3, 6, x6/m(el, x3), y6/x3): Res(7, (x = m(el, x))10
10, x7/x10, y7/m(el, x10)): (¬m(el,x) = z x = z)11
Res(4, 11, x4/el, x11/er, z11/el): (er = el)12
Res(1, 12,): ().

Ici, par exemple, Res(3, 6, x6/m(el, x3), y6/x3) signifie que dans la résolution de clause 3 avec clause 6, la variable x de la clause 6 est remplacée par m(el, x3) avec variable x de la clause 3. De manière analogue, y de la clause 6 est remplacé par x de clause 3.

Maintenant, nous voulons appliquer le démonstrateur E au problème. Les clauses sont transformées dans le langage de forme normale de clause LOP via le mappage

(¬A1 ··· ¬Am B1 ··· Bn) →B1;...;Bn < ¬A1,...,Am.

La syntaxe de LOP représente une extension de la syntaxe PROLOG (voir § 5) pour clauses non Horn. On obtient ainsi comme fichier d'entrée pour E

```
<- el = euh.                                  % requête
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X       % associativité de m
m(X,er) = X .                                 % élément neutre à gauche de m
. % élément neutre à droite de m
```

où l'égalité est modélisée par le symbole de prédicat gl. Appeler le prouveur délivre

```
unixprompt> epreuve halbgr1a.lop
# Statut du problème déterminé, construction d'un objet de preuve
# La preuve de l'état du problème commence
        0 : [--equal(el, er)] : initiale
        1 : [++equal(m(el,X1), X1)] : initiale
        2 : [++equal(m(X1,er), X1)] : initiale
        3 : [++equal(el,er)] : pm(2,1)
        4 : [--equal(el, el)] : rw(0,3)
        5 : [] : cn(4)
        6 : [] : 5 : {preuve}
# La preuve de l'état du problème se termine
```

Les littéraux positifs sont identifiés par ++ et les littéraux négatifs par --. Aux lignes 0 à 4, marqués d'une initiale, les clauses des données d'entrée sont à nouveau répertoriées. pm(a,b) représente une étape de résolution entre la clause a et la clause b. On voit que la preuve

trouvé par E est très similaire à la preuve créée manuellement. Parce que nous avons explicitement
modéliser l'égalité par le prédicat gl, les forces particulières de E ne viennent pas
en jeu. Maintenant, nous omettons les axiomes d'égalité et obtenons

```
<- el = euh.                                    % Requête
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X         % Assoziativ\"{a}t v. m
m(X,er) = X                .                    % liensneutrales El. v. m
                           .                    % rechtsneutrales El. v. m
```

comme entrée pour le prouveur.

La preuve devient également plus compacte. Nous voyons dans la sortie suivante du
prouver que la preuve consiste essentiellement en une seule étape d'inférence sur les deux clauses pertinentes 1 et
2.

```
unixprompt> epreuve halbgr1a.lop
# Statut du problème déterminé, construction d'un objet de preuve
# La preuve de l'état du problème commence
        0 : [--equal(el, er)] : initiale
        1 : [++equal(m(el,X1), X1)] : initiale
        2 : [++equal(m(X1,er), X1)] : initiale
        3 : [++equal(el,er)] : pm(2,1)
        4 : [--equal(el, el)] : rw(0,3)
        5 : : cn(4)
        6 :     []     : 5 : {preuve}
# La preuve de l'état du problème se termine
```

Le lecteur peut maintenant regarder de plus près les capacités de E (Exercice 3.9 sur
pages 55).

3.8 Candidatures

En mathématiques, les démonstrateurs de théorèmes automatisés sont utilisés pour certaines tâches spécialisées.
Par exemple, l'important théorème des quatre couleurs de la théorie des graphes a été prouvé pour la première fois en
1976 avec l'aide d'un étalon spécial. Cependant, les démonstrateurs automatisés jouent toujours un rôle mineur
rôle en mathématiques.

D'autre part, au début de l'IA, la logique des prédicats était d'une grande importance
pour le développement de systèmes experts dans des applications pratiques. En raison de ses problèmes
modélisation de l'incertitude (cf. § 4.4), les systèmes experts sont aujourd'hui le plus souvent développés
en utilisant d'autres formalismes.

Aujourd'hui, la logique joue un rôle de plus en plus important dans les tâches de vérification. Automatique
La vérification de programme est actuellement un domaine de recherche important entre l'IA et l'ingénierie logicielle.
Des systèmes logiciels de plus en plus complexes prennent désormais en charge les tâches

de plus en plus de responsabilité et de sécurité. Ici, une preuve de certaines caractéristiques de sécurité d'un programme est souhaitable. Une telle preuve ne peut être apportée par le test d'un programme fini, car en général il est impossible d'appliquer un programme à toutes les entrées possibles. C'est donc un domaine idéal pour les systèmes d'inférence généraux ou même spécialisés. Entre autres choses, on utilise aujourd'hui des protocoles cryptographiques dont les caractéristiques de sécurité ont été automatiquement vérifiées [FS97, Sch01]. Un autre défi pour l'utilisation de prouveurs automatisés est la synthèse du logiciel et du matériel. A cette fin, par exemple, les démonstrateurs doivent soutenir l'ingénieur logiciel dans la génération de programmes à partir de spécifications.

La réutilisation des logiciels est également d'une grande importance pour de nombreux programmeurs aujourd'hui. Le programmeur recherche un programme qui prend des données d'entrée avec certaines propriétés et calcule un résultat avec les propriétés souhaitées. Un algorithme de tri accepte les données d'entrée avec des entrées d'un certain type de données et à partir de celles-ci crée une permutation de ces entrées avec la propriété que chaque élément est inférieur ou égal à l'élément suivant.
Le programmeur formule d'abord une spécification de la requête en PL1 composée de deux parties. La première partie PREQ comprend les préconditions qui doivent être remplies avant que le programme souhaité ne soit appliqué. La deuxième partie POSTQ contient les postconditions, qui doivent tenir après l'application du programme souhaité.

Dans l'étape suivante, une base de données logicielle doit être recherchée pour les modules qui répondent à ces exigences. Pour vérifier cela formellement, la base de données doit stocker une description formelle des préconditions PREM et des postconditions POSTM pour chaque module M. Une hypothèse sur les capacités des modules est que les conditions préalables du module découlent des conditions préalables de la requête. ça doit tenir ça

$$\text{PREQ} \quad \text{PREM} .$$

Toutes les conditions requises comme pré-requis pour l'application du module M doivent apparaître comme pré-conditions dans la requête. Si, par exemple, un module de la base de données n'accepte que des listes d'entiers, alors les listes d'entiers en entrée doivent également apparaître comme conditions préalables dans la requête. Une exigence supplémentaire dans la requête selon laquelle, par exemple, seuls les nombres pairs apparaissent, ne pose pas de problème.

De plus, il doit être vrai pour les postconditions qui

$$\text{POSTM} \quad \text{POSTQ}.$$

Autrement dit, après l'application du module, tous les attributs requis par la requête doivent être remplis. Nous montrons maintenant l'application d'un démonstrateur de théorèmes à cette tâche dans un exemple de [Sch01].

Exemple 3.8 VDM-SL, le langage de spécification de la méthode de développement de Vienne, est souvent utilisé comme langage pour la spécification des pré- et post-conditions. Supposons que dans la base de données logicielle la description d'un module ROTATE est disponible, ce qui déplace le premier élément de la liste à la fin de la liste. Nous recherchons un module SHUFFLE, qui crée une permutation arbitraire de la liste. Les deux spécifications se lisent

ROTATE(l : Liste) l′ : Liste
pré vrai
poste
(l = [] l ′ = [])
(l = [] l ′ = (queue l)ˆ[tête l])

SHUFFLE(x : Liste) x′ : Liste
pré vrai
poste i : Article·
(x1, x2 : Liste · x = x1ˆ[i]ˆx2
y1, y2 : Liste · x ′ = y1ˆ[i]ˆy2)

"ˆ"
Ici représente la concaténation des listes, et "·" sépare les quantificateurs avec
leurs variables du reste de la formule. Les fonctions « head l » et « tail l »
choisissez respectivement le premier élément et le reste de la liste. La spécification
de SHUFFLE indique que chaque élément de liste i qui était dans la liste (x) avant le
l'application de SHUFFLE doit être dans le résultat (x ′) après la demande, et vice-versa
versa. Il faut maintenant montrer que la formule (PREQ PREM) (POSTM
POSTQ) est une conséquence de la base de connaissances contenant une description des
type de données Liste. Les deux spécifications VDM-SL donnent la tâche de preuve

l,l′ ,x,x′ : Liste · (l = x l ′ = x (w w))
(l = x l ′ =x ′ ((l = [] l ′ = []) (l = [] l ′ = (tl l)ˆ[hd l])
i : Item · (x1, x2 : Liste · x = x1ˆ[i]ˆx2 y1, y2 : Liste · x ′ = y1ˆ[i]ˆy2))),

qui peut ensuite être prouvé avec le démonstrateur SETHEO.

Dans les années à venir, le web sémantique représentera vraisemblablement une application importante de PL1. Le contenu du World Wide Web est censé devenir interprétable non seulement pour les personnes, mais pour les machines. À cette fin, des sites Web sont fournis avec une description de leur sémantique dans un langage de description formel. La recherche pour l'information sur le Web deviendra ainsi beaucoup plus efficace qu'aujourd'hui, où essentiellement seuls les blocs de construction de texte sont consultables.

Des sous-ensembles décidables de la logique des prédicats sont utilisés comme langages de description. Le développement de calculs efficaces pour le raisonnement est très important et étroitement lié aux langages de description. Une requête pour un futur moteur de recherche opérant sémantiquement pourrait lire (informellement) : Où en Suisse dimanche prochain à des altitudes sous 2000 mètres y aura-t-il du beau temps et des pistes de ski parfaitement préparées ? Pour répondre à une telle question, il faut un calcul capable de travailler très rapidement sur de grands ensembles de faits et de règles. Ici, les termes complexes des fonctions imbriquées sont moins importants.

En tant que cadre de description de base, le World Wide Web Consortium a développé le langage RDF (Resource Description Framework). S'appuyant sur RDF, le langage significativement plus puissant OWL (Web Ontology Language) permet la description de relations entre objets et classes d'objets, à l'instar de PL1 [SET09].
Les ontologies sont des descriptions de relations entre des objets possibles.

Une difficulté lors de la construction d'une description des innombrables sites Web est la dépense de travail et également la vérification de l'exactitude des descriptions sémantiques. Ici, les systèmes d'apprentissage automatique pour la génération automatique de descriptions peuvent être très utiles. Une utilisation intéressante de la génération « automatique » de sémantique dans le Web a été introduite par Luis von Ahn de l'Université Carnegie Mellon [vA06]. Il a développé des jeux informatiques dans lesquels les joueurs, répartis sur le réseau, sont censés décrire en collaboration des images avec des mots clés. Ainsi, les images se voient attribuer une sémantique de manière ludique et sans frais. Le lecteur peut tester les jeux et écouter un discours de l'inventeur dans l'exercice 3.10 à la page 55.

3.9 Résumé

Nous avons fourni les fondements, les termes et les procédures les plus importants de la logique des prédicats et nous avons montré que même l'une des tâches intellectuelles les plus difficiles, à savoir la démonstration de théorèmes mathématiques, peut être automatisée. Les démonstrateurs automatisés peuvent être employés non seulement en mathématiques, mais plutôt, en particulier, dans des tâches de vérification en informatique. Pour le raisonnement de tous les jours, cependant, la logique des prédicats dans la plupart des cas est mal adaptée. Dans les chapitres suivants et suivants, nous montrons ses points faibles et quelques alternatives modernes intéressantes. De plus, nous montrerons au Chap. 5 que l'on peut programmer élégamment avec la logique et ses extensions procédurales.

Toute personne intéressée par la logique du premier ordre, la résolution et d'autres calculs pour les démonstrateurs automatisés trouvera de bonnes instructions avancées dans [New00, Fit96, Bib82, Lov78, CL73]. Des références aux ressources Internet peuvent être trouvées sur le site Web de ce livre.

3.10 Exercices

Exercice 3.1 Soit le prédicat à trois places « enfant » et le prédicat à une place « femme homme » de l'exemple 3.2 à la page 35 . Définissez : (a) Un prédicat à une place "masculin". (b) Un prédicat à deux places "père" et "mère". (c) Un prédicat à deux places "frères et sœurs". (d) Un prédicat "parents(x, y, z)", qui est vrai si et seulement si x est le père et y est la mère de z.

(e) Un prédicat "oncle(x, y)", qui est vrai si et seulement si x est l'oncle de y (utilisez les prédicats déjà définis).

(f) Un prédicat à deux places "ancêtre" avec le sens : les ancêtres sont des parents, des grands-parents, etc. d'un nombre arbitraire de générations.

Exercice 3.2 Formalisez les énoncés suivants dans la logique des prédicats : (a) Toute personne a un père et une mère. (b) Certaines personnes ont des enfants. (c) Tous les oiseaux volent.

(d) Il y a un animal qui mange (des) animaux granivores. (e) Tout animal mange des plantes ou des animaux herbivores beaucoup plus petits que lui-même.

Exercice 3.3 Adaptez l'exercice 3.1 à la page 54 en utilisant des symboles de fonction à une place et l'égalité au lieu de « père » et « mère ».

Exercice 3.4 Donner des axiomes de la logique des prédicats pour la relation à deux places « < » comme ordre total. Pour une commande totale, nous devons avoir (1) Deux éléments quelconques sont comparables. (2) Il est symétrique. (3) Il est transitif.

Exercice 3.5 Unifiez (si possible) les termes suivants et donnez le MGU et les termes résultants. (a) p(x, f (y)), p(f (z), u) (b) p(x, f (x)), p(y, y) (c) x = 4 − 7 · x, cos y = z (d) x < 2 · x, 3 < 6 (e) q(f (x, y, z), f (g(w, w), g(x, x), g(y, y))), q(u, u)

Exercice 3.6 (a)

Transformez le paradoxe de Russell de l'exemple 3.7 à la page 45 en CNF. (b) Montrez que la clause vide ne peut pas être dérivée en utilisant la résolution sans factorisation de (3.7) à la page 45. Essayez de comprendre cela intuitivement.

Exercice 3.7 (a)

Pourquoi la résolution avec l'ensemble des stratégies de support est-elle incomplète ? (b) Justifier (sans prouver) pourquoi l'ensemble de la stratégie de soutien devient complet si (KB ¬Q)\SOS est satisfaisable. (c) Pourquoi la résolution avec la règle littérale pure est-elle complète ?

Exercice 3.8 Formaliser et prouver avec résolution que dans un semi-groupe avec au moins deux éléments différents a,b, un élément neutre à gauche e, et un élément nul à gauche n, ces deux éléments doivent être différents, c'est-à-dire que n = e. Utilisez la démodulation, qui permet de remplacer « comme par comme ».

Exercice 3.9 Obtenir le démonstrateur de théorème E [Sch02] ou un autre démonstrateur et prouver les énoncés suivants. Comparez ces preuves avec celles du texte. (a) L'affirmation de l'exemple 2.3 à la page 24. (b) Le paradoxe de Russell de l'exemple 3.7 à la page 45. (c) L'affirmation de l'exercice 3.8.

Exercice 3.10 Testez les jeux www.espgame.org et www.peekaboom.org, qui aident à attribuer une sémantique aux images sur le Web. Écoutez la conférence sur le calcul humain par Luis von Ahn sur : http://video.google.de/videoplay?docid= -8246463980976635143.

Limites de la logique

4

4.1 Le problème de l'espace de recherche

Comme déjà mentionné à plusieurs endroits, dans la recherche d'une preuve, il existe presque toujours de nombreuses (selon le calcul, potentiellement une infinité) de possibilités d'application de règles d'inférence à chaque étape. Le résultat est la croissance explosive susmentionnée de l'espace de recherche (Fig. 4.1 à la page 58). Dans le pire des cas, toutes ces possibilités doivent être essayées afin de trouver la preuve, ce qui n'est généralement pas possible dans un délai raisonnable.

Si nous comparons des démonstrateurs automatisés ou des systèmes d'inférence avec des mathématiciens ou des experts humains qui ont de l'expérience dans des domaines particuliers, nous faisons des observations intéressantes. D'une part, les mathématiciens expérimentés peuvent prouver des théorèmes qui sont hors de portée des démonstrateurs automatisés. D'autre part, les démonstrateurs automatisés effectuent des dizaines de milliers d'inférences par seconde. Un humain, en revanche, effectue peut-être une inférence par seconde. Bien que les experts humains soient beaucoup plus lents au niveau de l'objet (c'est-à-dire pour effectuer des inférences), ils résolvent apparemment les problèmes difficiles beaucoup plus rapidement.

Il y a plusieurs raisons à cela. Nous, les humains, utilisons des calculs intuitifs qui fonctionnent à un niveau supérieur et effectuent souvent de nombreuses inférences simples d'un démonstrateur automatisé en une seule étape. De plus, nous utilisons des lemmes, c'est-à-dire des formules vraies dérivées que nous connaissons déjà et n'avons donc pas besoin de les prouver à nouveau à chaque fois. Entre-temps, il existe également des prouveurs automatiques qui travaillent avec de telles méthodes. Mais même eux ne peuvent pas encore rivaliser avec les experts humains.

Un autre avantage beaucoup plus important de nous, les humains, est l'intuition, sans laquelle nous ne pourrions résoudre aucun problème difficile [PS09]. La tentative de formalisation de l'intuition pose problème. L'expérience des projets d'IA appliquée montre que dans des domaines complexes comme la médecine (voir section 7.3) ou les mathématiques, la plupart des experts sont incapables de formuler verbalement ces méta-savoirs intuitifs, encore moins de les formaliser. On ne peut donc pas programmer ces connaissances ni les intégrer dans des calculs sous forme d' heuristiques. Les heuristiques sont des méthodes qui, dans de nombreux cas, peuvent grandement simplifier ou raccourcir le chemin vers l'objectif, mais dans certains cas (généralement rarement) peuvent considérablement allonger le chemin.

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique,
DOI 10.1007/978-0-85729-299-5_4,

Fig. 4.1 Conséquences possibles de l'explosion d'un espace de recherche

au but. La recherche heuristique est importante non seulement pour la logique, mais généralement pour la résolution de problèmes en IA et sera donc traitée en détail au Chap. 6.

Une approche intéressante, poursuivie depuis environ 1990, est l'application de techniques d'apprentissage automatique à l'apprentissage d'heuristiques pour diriger la recherche de systèmes d'inférence, que nous allons brièvement esquisser maintenant. Un prouveur de résolution a, lors de la recherche d'une preuve, des centaines ou plus de possibilités d'étapes de résolution à chaque étape, mais seules quelques-unes mènent au but. L'idéal serait que le prouveur puisse demander à un oracle quelles sont les deux clauses qu'il doit utiliser à l'étape suivante pour trouver rapidement la preuve. Il existe des tentatives pour construire de tels modules de direction de preuve, qui évaluent les différentes alternatives pour l'étape suivante, puis choisissent l'alternative avec la meilleure note. Dans le cas de la résolution, la notation des clauses disponibles pourrait être calculée par une fonction qui calcule une valeur basée sur le nombre de littéraux positifs, la complexité des termes, etc., pour chaque paire de clauses résolubles.

Comment cette fonction peut-elle être implémentée ? Parce que cette connaissance est « intuitive », le programmeur ne la connaît pas. Au lieu de cela, on essaie de copier la nature et utilise des algorithmes d'apprentissage automatique pour apprendre à partir de preuves réussies [ESS89, SE90]. Les attributs de toutes les paires de clauses participant aux étapes de résolution réussies sont stockés comme positifs, et les attributs de toutes les résolutions infructueuses sont stockés comme négatifs.
Ensuite, en utilisant ces données d'apprentissage et un système d'apprentissage automatique, un programme est généré qui peut évaluer les paires de clauses de manière heuristique (voir la section 9.5).

Une approche différente et plus efficace pour améliorer le raisonnement mathématique consiste à suivie de systèmes interactifs qui fonctionnent sous le contrôle de l'utilisateur. Ici un pourrait nommer des programmes de calcul formel tels que Mathematica, Maple ou Maxima, qui peut effectuer automatiquement des manipulations mathématiques symboliques difficiles. La recherche de la preuve, cependant, est entièrement laissée à l'humain. Le prouveur interactif mentionné ci-dessus Isabelle [NPW02] fournit nettement plus de support lors de la preuve recherche. Il existe actuellement plusieurs projets, comme Omega [SB04] et MKM,1 pour le développement de systèmes d'aide aux mathématiciens lors des preuves.

En résumé, on peut dire qu'en raison du problème d'espace de recherche, l'automatisation aujourd'hui, les démonstrateurs ne peuvent prouver que des théorèmes relativement simples dans des domaines particuliers avec quelques axiomes.

4.2 Décidabilité et incomplétude

La logique des prédicats du premier ordre fournit un outil puissant pour la représentation des connaissances et du raisonnement. Nous savons qu'il existe des démonstrateurs de calculs et de théorèmes corrects et complets. Lorsqu'il prouve un théorème, c'est-à-dire un énoncé vrai, un tel démonstrateur est très utile car, en raison de l'exhaustivité, on sait après un temps fini que l'énoncé est vraiment vrai. Et si l'énoncé n'est pas vrai ? Le théorème de complétude (Théorème 3.3 page 41) ne répond pas à cette question2. Plus précisément, il n'y a pas de processus qui peut prouver ou réfuter n'importe quelle formule de PL1 en temps fini, car elle soutient que

> Théorème 4.1 L'ensemble des formules valides en logique des prédicats du premier ordre est semi décidable.

Ce théorème implique qu'il existe des programmes (prouveurs de théorèmes) qui, étant donné un formule vraie (valide) en entrée, déterminez sa vérité en un temps fini. Si la formule n'est pas valide, cependant, il peut arriver que le démonstrateur ne s'arrête jamais. (Le lecteur pourra aborder cette question dans l'exercice 4.1 , page 65.) La logique propositionnelle est décidable car la méthode des tables de vérité fournit tous les modèles d'une formule en temps fini. Évidemment, la logique des prédicats avec des quantificateurs et des symboles de fonction imbriqués est un langage un peu trop puissant pour être décidable.

D'un autre côté, la logique des prédicats n'est pas assez puissante pour de nombreux objectifs. On souhaite souvent faire des déclarations sur des ensembles de prédicats ou de fonctions. Ce ne fonctionne pas dans PL1 car il ne connaît que les quantificateurs pour les variables, mais pas pour prédicats ou fonctions.

Kurt Gödel a montré, peu de temps après son théorème de complétude pour PL1, que la complétude est perdue si nous étendons PL1 même minimalement pour construire une logique d'ordre supérieur. Une logique du premier ordre ne peut quantifier que sur des variables. Une logique du second ordre peut aussi

1www.mathweb.org/mathweb/demo.html.

2Ce cas est particulièrement important dans la pratique, car si je sais déjà qu'une affirmation est vraie, Je n'ai plus besoin de prouveur.

quantifier sur des formules du premier ordre, et une logique du troisième ordre peut quantifier sur des formules du second ordre. Même en ajoutant seulement l'axiome d'induction pour les nombres naturels, la logique est incomplète. L'énoncé "Si un prédicat p(n) est vrai pour n, alors p(n + 1) est aussi vrai", ou

$$pp(n) \quad p(n+1)$$

est une proposition du second ordre car elle quantifie sur un prédicat. Gödel a démontré le théorème suivant :

Théorème 4.2 (Théorème d'incomplétude de Gödel) Tout système d'axiomes pour les nombres naturels avec addition et multiplication (arithmétique) est incomplet.
C'est-à-dire qu'il y a des déclarations vraies en arithmétique qui ne sont pas prouvables.

La preuve de Gödel fonctionne avec ce qu'on appelle la gödelisation, dans laquelle chaque formule arithmétique est codée sous forme de nombre. Il obtient un numéro Gödel unique. La gödelisation est maintenant utilisée pour formuler la proposition

F = "Je ne suis pas démontrable."

dans le langage de l'arithmétique. Cette formule est vraie pour la raison suivante. Supposons que F est faux. On peut alors prouver F et donc montrer que F n'est pas démontrable. C'est une contradiction. Donc F est vrai et donc non démontrable.

L'arrière-plan plus profond de ce théorème est que les théories mathématiques (systèmes d'axiomes) et, plus généralement, les langages deviennent incomplets si le langage devient trop puissant. Un exemple similaire est la théorie des ensembles. Ce langage est si puissant qu'on peut formuler des paradoxes avec lui. Ce sont des affirmations qui se contredisent, comme celle que nous connaissons déjà de l'exemple 3.7 page 45 sur les barbiers qui rasent tous ceux qui ne se rasent pas eux-mêmes (voir exercice 4.2 page 65) 3 . suffisamment puissants pour décrire les mathématiques et des applications intéressantes, nous faisons passer clandestinement des contradictions et des incomplétude par la porte dérobée. Cela ne signifie pas pour autant que les logiques d'ordre supérieur sont totalement inadaptées aux méthodes formelles. Il existe certainement des systèmes formels ainsi que des démonstrateurs pour les logiques d'ordre supérieur.

4.3 Le pingouin volant

Avec un exemple simple, nous démontrerons un problème fondamental de logique et des approches de solutions possibles. Au vu des déclarations

3De nombreux autres paradoxes logiques peuvent être trouvés dans [Wie].

Fig. 4.2 Le pingouin volant Titi

1. Titi est un pingouin 2. Les
pingouins sont des oiseaux
3. Les oiseaux
peuvent voler Formalisé en PL1, la base de connaissances KB donne :

$$\text{pingouin}(\text{titi})$$
$$\text{pingouin}(x) \quad \text{oiseau}(x)$$
$$\text{oiseau}(x) \quad \text{voler}(x)$$

À partir de là (par exemple avec résolution), fly(tweety) peut être dérivé (Fig. 4.2).4 Évidemment, la formalisation des attributs de vol des manchots est insuffisante. Nous essayons la déclaration supplémentaire que les pingouins ne peuvent pas voler, c'est-à-dire

$$\text{pingouin}(x) \quad \neg\text{mouche}(x)$$

De là, ¬fly(tweety) peut être dérivé. Mais voler (tweety) est toujours vrai. La base de connaissances est donc incohérente. Nous remarquons ici une caractéristique importante de la logique, à savoir la monotonie. Bien que nous déclarions explicitement que les pingouins ne peuvent pas voler, le site opposé peut toujours être dérivé.

Définition 4.1 Une logique est dite monotone si, pour une base de connaissances arbitraire KB et une formule arbitraire φ , l'ensemble des formules dérivables de KB est un sous-ensemble des formules dérivables de KB φ .

4L'exécution formelle de cette preuve simple et de la suivante peut être laissée au lecteur (Exercice 4.3 page 65).

Si un ensemble de formules est étendu, alors, après l'extension, toutes les déclarations précédemment dérivables peuvent encore être prouvées, et des déclarations supplémentaires peuvent potentiellement être également prouvées. L'ensemble des énoncés prouvables croît donc de manière monotone lorsque l'ensemble des formules est étendu. Pour notre exemple, cela signifie que l'extension de la base de connaissances ne conduira jamais à notre objectif. Nous modifions donc KB en remplaçant l'énoncé manifestement faux « (tous) les oiseaux peuvent voler » par l'énoncé plus exact « (tous) les oiseaux sauf les pingouins peuvent voler » et obtenons comme KB2 les clauses suivantes :

pingouin(titi)
pingouin(x) oiseau(x)
oiseau(x) ¬pingouin(x) mouche(x)
pingouin(x) ¬mouche(x)

Maintenant, le monde est apparemment de nouveau en ordre. Nous pouvons dériver ¬fly(tweety), mais pas fly(tweety), car pour cela nous aurions besoin de ¬penguin(x), qui n'est cependant pas dérivable. Tant qu'il n'y a que des pingouins dans ce monde, la paix règne. Cependant, tout oiseau normal cause immédiatement des problèmes. Nous souhaitons ajouter le corbeau Abraxas (du livre allemand "La Petite Sorcière") et obtenir

corbeau(abraxas)
corbeau(x) oiseau(x)
pingouin(tweety)
pingouin(x) oiseau(x)
oiseau(x) ¬pingouin(x) mouche(x)
pingouin(x) ¬mouche(x)

Nous ne pouvons rien dire sur les attributs de vol d'Abraxas car nous avons oublié de formuler que les corbeaux ne sont pas des pingouins. Ainsi nous étendons KB3 à KB4 :

corbeau(abraxas)
corbeau(x) oiseau(x)
corbeau(x) ¬pingouin(x)
pingouin(tweety)
pingouin(x) oiseau(x)
oiseau(x) ¬pingouin(x) voler(x)
pingouin(x) ¬mouche(x)

Le fait que les corbeaux ne sont pas des pingouins, ce qui va de soi pour les humains, doit être explicitement ajouté ici. Pour la construction d'une base de connaissances avec l'ensemble des quelque 9 800 types d'oiseaux dans le monde, il faut donc préciser pour chaque type d'oiseau (sauf les manchots) qu'il n'appartient pas aux manchots. Il faut procéder de manière analogue pour toutes les autres exceptions comme l'autruche.

Pour chaque objet de la base de connaissances, en plus de ses attributs, tous les les attributs qu'il n'a pas doivent être listés.

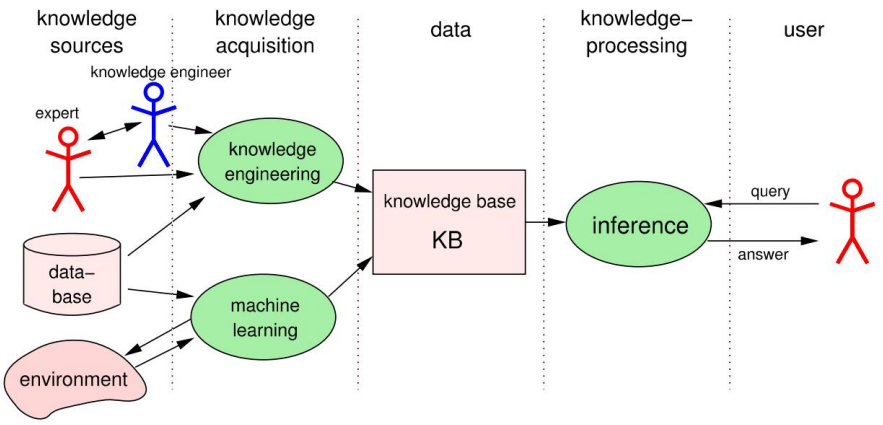


Fig. 1.6 Structure d'un système classique de traitement des connaissances

plus facile de remplacer la base de connaissances d'un système expert médical que de programmer un tout nouveau système.

Grâce au découplage de la base de connaissances de l'inférence, la connaissance peut être stocké de manière déclarative. Dans la base de connaissances, il n'y a qu'une description des connaissances indépendantes du système d'inférence utilisé. Sans cela une séparation claire, la connaissance et le traitement des étapes d'inférence seraient imbriqués, et toute modification des connaissances serait très coûteuse.

Le langage formel en tant qu'interface pratique entre l'homme et la machine se prête à la représentation des connaissances dans la base de connaissances. Dans les chapitres suivants, nous apprendrons à connaître toute une série de ces langages. D'abord, aux chap. 2 et 3 il y a le calcul propositionnel et la logique des prédicats du premier ordre (PL1). Mais d'autres formalismes tels que la logique probabiliste, la logique floue ou les arbres de décision sont également présentés. Nous commençons par le calcul propositionnel et les systèmes d'inférence associés. Bâtir sur cela, nous présenterons la logique des prédicats, un langage puissant accessible par les machines et très important en IA.

Comme exemple pour un système basé sur la connaissance à grande échelle, nous voulons nous référer au agent logiciel "Watson". Développé chez IBM en collaboration avec un certain nombre d'universités, Watson est un programme de questions-réponses, qui peut être alimenté avec des indices donnés en naturel langue. Il fonctionne sur une base de connaissances comprenant quatre téraoctets de stockage sur disque dur, y compris le texte intégral de Wikipédia [FNA+09]. Watson a été développé au sein Le projet DeepQA d'IBM qui est caractérisé dans [Dee11] comme suit :

> Le projet DeepQA d'IBM façonne un grand défi en informatique qui vise à illustrent comment l'accessibilité large et croissante du contenu en langage naturel et l'intégration et l'avancement du traitement du langage naturel, de la recherche d'informations, de l'apprentissage automatique, de la représentation et du raisonnement des connaissances et du calcul massivement parallèle peuvent conduire la technologie de réponse automatique aux questions à domaine ouvert vers un point où il rivalise clairement et systématiquement avec la meilleure performance humaine.

Dans l'émission de quiz télévisée américaine "Jeopardy!", En février 2011, Watson a battu les deux champions humains Brad Rutter et Ken Jennings dans un match à deux, match à points combinés et a remporté le prix d'un million de dollars. L'un des partenaires de Watson

Ses points forts particuliers étaient sa réaction très rapide aux questions avec pour résultat que Watson appuyait souvent sur le buzzer (à l'aide d'un solénoïde) plus rapidement que ses concurrents humains et était alors capable de donner la première réponse à la question.

Les hautes performances et les temps de réaction courts de Watson étaient dus à une implémentation sur 90 serveurs IBM Power 750, chacun contenant 32 processeurs, soit 2880 processeurs parallèles.

1.5 Des exercices

Exercice 1.1 Testez quelques-uns des chatterbots disponibles sur Internet. Commencez par exemple par www.hs-weingarten.de/~ertel/aibook dans la collection de liens sous Tur ingtest/Chatterbots, ou par www.simonlaven.com ou www.alicebot.org. Écrivez une question de départ et mesurez le temps qu'il faut, pour chacun des différents programmes, jusqu'à ce que vous sachiez avec certitude qu'il ne s'agit pas d'un humain.

Exercice 1.2 Sur www.pandorabots.com, vous trouverez un serveur sur lequel vous pouvez construire un chatterbot avec le langage de balisage AIML assez facilement. Selon votre niveau d'intérêt, développez un chatterbot simple ou complexe, ou modifiez-en un existant.

Exercice 1.3 Donnez les raisons de l'inadéquation du test de Turing comme définition de «l'intelligence artificielle» dans l'IA pratique.

Exercice 1.4 De nombreux processus d'inférence, d'apprentissage, etc. bien connus sont NP-complet ou même indécidable. Qu'est-ce que cela signifie pour l'IA ?

Exercice 1.5 (a)

Pourquoi un agent déterministe à mémoire n'est-il pas une fonction de l'ensemble de toutes les entrées vers l'ensemble de toutes les sorties, au sens mathématique ? (b)

Comment peut-on changer l'agent avec mémoire, ou le modéliser, de telle sorte qu'il devienne équivalent à une fonction mais ne perd pas sa mémoire ?

Exercice 1.6 Soit un agent avec de la mémoire qui peut se déplacer dans un plan. De ses capteurs, il reçoit à coups d'horloge d'un intervalle régulier t sa position exacte (x, y) en coordonnées cartésiennes.
(a) Donnez une formule avec

laquelle l'agent peut calculer sa vitesse à partir du courant temps t et la mesure précédente de t − t.

(b) Comment doit-on changer l'agent pour qu'il puisse aussi calculer son accélération ? Fournissez une formule ici aussi.

Exercice 1.7

(a) Déterminez pour les deux agents de l'exemple 1.1 à la page 11 les coûts engendrés par les erreurs et comparez les résultats. Supposons ici que devoir supprimer manuellement un courrier indésirable coûte un centime et que récupérer un courrier électronique supprimé, ou la perte d'un courrier électronique, coûte un dollar.

(b) Déterminez pour les deux agents le profit créé par des classifications correctes et comparez les résultats. Supposons que pour chaque e-mail souhaité reconnu, un bénéfice d'un dollar s'accumule et pour chaque e-mail de spam correctement supprimé, un bénéfice d'un centime.

Logique propositionnelle 2

Dans la logique propositionnelle, comme son nom l'indique, les propositions sont reliées par des opérateurs logiques. L'énoncé « la rue est mouillée » est une proposition, tout comme « il pleut ».
Ces deux propositions peuvent être reliées pour former la nouvelle proposition

s'il pleut, la rue est mouillée.

Écrit plus formellement

il pleut la rue est mouillée.

Cette notation a l'avantage que les propositions élémentaires apparaissent à nouveau sous une forme non altérée. Afin que nous puissions travailler précisément avec la logique propositionnelle, nous commencerons par une définition de l'ensemble de toutes les formules de la logique propositionnelle.

2.1 Syntaxe

Définition 2.1 Soient Op = {¬, , , , , (,)} l'ensemble des opérateurs logiques et Σ un ensemble de symboles. Les ensembles Op, Σ et {t,f } sont deux à deux disjoints. Σ est appelé la signature et ses éléments sont les variables de proposition. L'ensemble des formules de la logique propositionnelle est maintenant défini récursivement : • t et f sont des formules (atomiques). • Toutes les variables de proposition, c'est-à-dire tous les éléments de Σ, sont de formule (atomique) Las.

• Si A et B sont des formules, alors ¬A, (A), A B, A B, A B, A B sont aussi des formules.

Cette élégante définition récursive de l'ensemble de toutes les formules nous permet de générer une infinité de formules. Par exemple, étant donné Σ = {A,B,C},

UNE B, UNE B C, UNE UNE A, C B A, (¬A B) (¬C A)

sont des formules. (((A)) B) est également une formule syntaxiquement correcte.

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique, DOI
10.1007/978-0-85729-299-5_2,

Définition 2.2 On lit les symboles et opérateurs de la manière suivante :

t : "vrai"

f : "faux"

¬A : "pas A" (négation)

A B : "A et B" (conjonction)

A B : "A ou B" (disjonction)

A B : "si A alors B" (implication (également appelée implication matérielle))

A B : « A si et seulement si B » (équivalence)

Les formules ainsi définies sont jusqu'ici des constructions purement syntaxiques sans signification. Il nous manque encore la sémantique.

2.2 Sémantique

En logique propositionnelle, il existe deux valeurs de vérité : t pour « vrai » et f pour « faux ». Nous commencer par un exemple et se demander si la formule A B est vraie. Le réponse est : cela dépend si les variables A et B sont vraies. Par exemple, si A signifie "Il pleut aujourd'hui" et B pour "Il fait froid aujourd'hui" et ce sont tous les deux vrais, alors A B est vraie. Si, au contraire, B représente "Il fait chaud aujourd'hui" (et c'est faux), alors A B est faux.

Il faut évidemment assigner des valeurs de vérité qui reflètent l'état du monde à propo variables de position. C'est pourquoi nous définissons

Définition 2.3 Une application I : Σ → {w, f }, qui attribue une valeur de vérité à chaque variable de proposition est appelée une interprétation.

Parce que chaque variable de proposition peut prendre deux valeurs de vérité, chaque formule de logique propositionnelle avec n variables différentes a 2n interprétations différentes. Nous définir les valeurs de vérité pour les opérations de base en montrant toutes les interprétations possibles dans une table de vérité (voir tableau 2.1 à la page 17).

La formule vide est vraie pour toutes les interprétations. Afin de déterminer la vérité valeur pour les formules complexes, il faut aussi définir l'ordre des opérations pour les logiques les opérateurs. Si les expressions sont entre parenthèses, le terme entre parenthèses est évalué

Tableau 2.1 Définition de la opérateurs logiques par vérité tableau

AB (A) ¬AA	BA	BA	BA	B		
tt ft				t	t	t
t ft ff				t	F	F
pi pi		F		t	t	F
fff t		F		F	t	t

d'abord. Pour les formules sans parenthèses, les priorités sont classées comme suit, en commençant avec la liaison la plus forte : ¬, , , , .

Pour différencier clairement l'équivalence des formules et l'équivalence syntaxique alence, nous définissons

Définition 2.4 Deux formules F et G sont dites sémantiquement équivalentes si ils prennent la même valeur de vérité pour toutes les interprétations. On écrit F ≡ G.

L'équivalence sémantique sert avant tout à pouvoir utiliser le métalangage, qui est, langage naturel, de parler du langage objet, à savoir la logique. La déclaration « A ≡ B » signifie que les deux formules A et B sont sémantiquement équivalentes. Le l'énoncé "A B" est en revanche un objet syntaxique du langage formel de la logique propositionnelle.

Selon le nombre d'interprétations dans lesquelles une formule est vraie, nous pouvons diviser formules dans les classes suivantes :

Définition 2.5 Une formule est appelée

- Satisfiable s'il est vrai pour au moins une interprétation.
- Logiquement valide ou simplement valide si elle est vraie pour toutes les interprétations. Vrai car les mulas sont aussi appelées tautologies.
- Insatisfiable s'il n'est vrai pour aucune interprétation.

Chaque interprétation qui satisfait une formule est appelée un modèle de la formule.

Il est clair que la négation de toute formule généralement valide est insatisfaisante. Le néga tion d'une formule satisfiable, mais pas généralement valide F est satisfiable.

Nous sommes maintenant capables de créer des tables de vérité pour des formules complexes afin de déterminer leur valeurs de vérité. Nous mettons cela en action immédiatement en utilisant des équivalences de formules qui sont importants dans la pratique.

Théorème 2.1 Les opérations , sont commutatives et associatives, et les
les équivalences suivantes sont généralement valables :

¬A B UNE B	(implication)
UNE B ¬B ¬A	(contraposition)
(UNE B) (B UNE) (A B) ¬(A B)	(équivalence)
¬A ¬B ¬(A B) ¬A ¬B	(loi de De Morgan)

A (B C) (A B) (A C) (loi distributive)
UNE (B C) (A B) (A C)

UNE ¬A w	(tautologie)
UNE ¬A f	(contradiction)

UNE f UNE
UNE w w

UNE f f
UNE w UNE

Preuve Pour montrer la première équivalence, on calcule la table de vérité pour ¬A B et
A B et voyez que les valeurs de vérité pour les deux formules sont les mêmes pour toutes les
interprétations. Les formules sont donc équivalentes, et donc toutes les valeurs du dernier
colonne sont des "t".

AB ¬A ¬A BA	B (¬A	B)	(A	B)
ttf tt ff f ft t fft t		t		t
		F		t
	t	t		t
		t		t

Les preuves pour les autres équivalences sont similaires et sont recommandées comme exercices
pour le lecteur (Exercice 2.2 à la page 29).

2.3 Systèmes de preuve

En IA, nous sommes intéressés à prendre des connaissances existantes et à en tirer de nouvelles
connaissances ou répondre à des questions. En logique propositionnelle, cela revient à montrer que
une base de connaissances KB—c'est-à-dire une formule de logique propositionnelle (éventuellement étendue)—
une formule Q1 suit. Ainsi, nous définissons d'abord le terme « implication ».

1Ici, Q signifie requête.

Définition 2.6 Une formule KB entraîne une formule Q (ou Q découle de KB) si tout modèle de KB est aussi un modèle de Q. On écrit KB | Q

En d'autres termes, dans chaque interprétation dans laquelle KB est vrai, Q est également vrai. Plus succinctement, chaque fois que KB est vrai, Q est également vrai. Parce que, pour le concept d'implication, on fait intervenir des interprétations de variables, on a affaire à un concept sémantique.

Toute formule non valide choisit pour ainsi dire un sous-ensemble de l'ensemble de toutes les interprétations comme modèle. Les tautologies telles que A ¬A, par exemple, ne restreignent pas le nombre d'interprétations satisfaisantes car leur proposition est vide. La formule vide est donc vraie dans toutes les interprétations. Pour toute tautologie T alors | T Intuitivement cela signifie que les tautologies sont toujours vraies, sans restriction des interprétations par une formule. Pour faire court, nous écrivons | T. Nous montrons maintenant un lien important entre le concept sémantique d'implication et l'implication syntaxique.

Théorème 2.2 (Théorème de déductibilité)

Un | B si et seulement si | A B.

Preuve Observez la table de vérité pour l'implication :

ABA	B
ttt	
t ff ft t fft	

Une implication arbitraire A B est évidemment toujours vraie sauf avec l'interprétation A → t,B → f . Supposons que A | B tient. Cela signifie que pour chaque interprétation qui rend A vrai, B est également vrai. La deuxième ligne critique de la table de vérité ne s'applique même pas dans ce cas. Donc A B est vrai, ce qui signifie que A B est une tautologie. Ainsi, une direction de l'énoncé a été montrée.

Supposons maintenant que A B est vérifié. Ainsi, la deuxième ligne critique de la table de vérité est également en lock-out. Tout modèle de A est alors aussi un modèle de B. Alors A | B tient.

Si l'on veut montrer que KB entraîne Q, on peut aussi démontrer au moyen de la méthode des tables de vérité que KB Q est une tautologie. Ainsi, nous avons notre premier système de preuve pour la logique propositionnelle, qui est facilement automatisé. L'inconvénient de cette méthode est le temps de calcul très long dans le pire des cas. Plus précisément, dans le pire des cas avec n variables de proposition, pour toutes les 2n interprétations des variables, la formule KB Q doit être évaluée. Le temps de calcul croît donc de manière exponentielle

avec le nombre de variables. Par conséquent, ce processus est inutilisable pour de grands nombres de variables, du moins dans le pire des cas.

Si une formule KB entraîne une formule Q, alors par le théorème de déduction KB Q est une tautologie. Donc la négation ¬(KB Q) est insatisfiable. Nous avons

¬(KB Q) ≡ ¬(¬KB Q) ≡ KB ¬Q.

Donc KB ¬Q est aussi satisfiable. Nous formulons cette conséquence simple mais importante du théorème de déduction sous forme de théorème.

Théorème 2.3 (Preuve par contradiction) KB | Q si et seulement si KB ¬Q est insatisfiable.

Pour montrer que la requête Q découle de la base de connaissances KB, nous pouvons également ajouter la requête négative ¬Q à la base de connaissances et en déduire une contradiction. En raison de l'équivalence A ¬A f du théorème 2.1 à la page 18 , nous savons qu'une contradiction est insatisfaisante. Par conséquent, Q est prouvé. Cette procédure, fréquemment utilisée en mathématiques, est également utilisée dans divers calculs de preuve automatiques tels que le calcul de résolution et dans le traitement des programmes PROLOG.

Une façon d'éviter d'avoir à tester toutes les interprétations avec la méthode des tables de vérité est la manipulation syntaxique des formules KB et Q par application de règles d'inférence dans le but de les simplifier grandement, de sorte qu'au final on voit instantanément que KB | Q. Nous appelons ce processus syntaxique dérivation et écrivons KB Q. De tels systèmes de preuve syntaxique sont appelés calculi. Pour s'assurer qu'un calcul ne génère pas d'erreurs, nous définissons deux propriétés fondamentales des calculs.

Définition 2.7 Un calcul est dit sain si toute proposition dérivée suit sémantiquement. Autrement dit, s'il est vrai pour les formules KB et Q que

si KB Q alors KB | Q

Un calcul est dit complet si toutes les conséquences sémantiques peuvent en être dérivées. C'est-à-dire que pour les formules KB et Q, il contient :

si Ko | Q puis KB Q.

La justesse d'un calcul assure que toutes les formules dérivées sont en fait des conséquences sémantiques de la base de connaissances. Le calcul ne produit pas de "fausses conséquences". L'exhaustivité d'un calcul, d'autre part, garantit que le calcul ne néglige rien. Un calcul complet trouve toujours une preuve si la formule à prouver découle de la base de connaissances. Si un calcul est bon

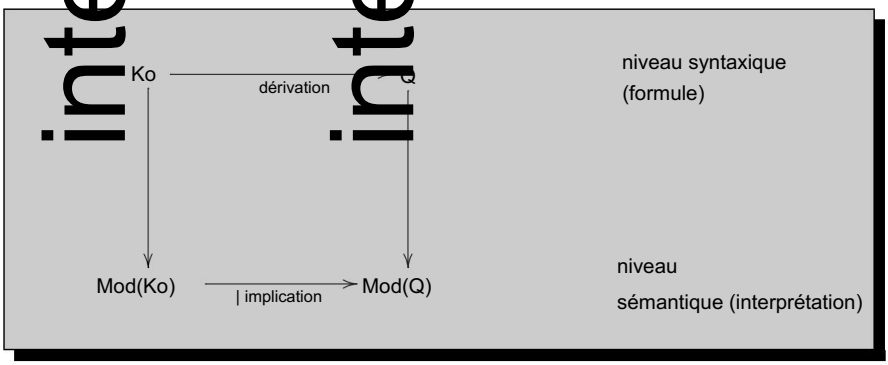


Fig. 2.1 Dérivation syntaxique et implication sémantique. Mod(X) représente l'ensemble des modèles d'une formule X

et complète, alors dérivation syntaxique et implication sémantique sont deux relations équivalentes (voir Fig. 2.1).

Pour garder les systèmes de preuve automatique aussi simples que possible, ceux-ci sont généralement fabriqués pour opérer sur des formules en forme normale conjonctive.

Définition 2.8 Une formule est en forme normale conjonctive (CNF) si et seulement si elle consiste en une conjonction

$$K1 \quad K2 \quad \cdots \quad Km$$

de clauses. Une clause Ki consiste en une disjonction

$$(Li1 \quad Li2 \quad \cdots \quad Lini)$$

de littéraux. Enfin, un littéral est une variable (littéral positif) ou une variable niée (littéral négatif).

La formule (A B ¬C) (A B) (¬B ¬C) est sous forme normale conjonctive. La forme normale conjonctive n'impose pas de restriction sur l'ensemble des formules car :

Théorème 2.4 Toute formule logique propositionnelle peut être transformée en une forme normale conjonctive équivalente.

Exemple 2.1 On met A B C D sous forme normale conjonctive en utilisant les équivalences du Théorème 2.1 page 18 :

UNE B C
D ≡ ¬(A B) (C D) (implication) ≡ (¬A ¬B) (C D) (de Morgan) ≡
(¬A (C D)) (¬B (C D)) (loi distributive) ≡ ((¬A C) (¬A
D)) ((¬B C) (¬B D)) (loi distributive) ≡ (¬A C) (¬A D) (¬B
C) (¬B D) (loi associative)

Il ne nous manque plus qu'un calcul pour la preuve syntaxique des formules de la logique propositionnelle. Nous commençons par le modus ponens, une règle d'inférence simple et intuitive, qui, à partir de la validité de A et A B, permet la dérivation de B. Nous l'écrivons formellement comme

$$\frac{A,\ A \quad B}{B}.$$

Cette notation signifie que nous pouvons dériver la ou les formules sous la ligne à partir des formules séparées par des virgules au-dessus de la ligne. Modus ponens en règle générale par lui-même, bien que valable, n'est pas complet. Si nous ajoutons des règles supplémentaires, nous pouvons créer un calcul complet, que nous ne souhaitons cependant pas considérer ici. Au lieu de cela, nous allons étudier la règle de résolution

$$\frac{\text{UNE} \quad B,\ \neg B \quad C}{A \quad C} \tag{2.1}$$

comme alternative. La clause dérivée est appelée résolvante. Par une simple transformation on obtient la forme équivalente

$$\frac{\text{UNE} \quad B,\ B \quad C}{A \quad C}.$$

Si on fixe A à f , on voit que la règle de résolution est une généralisation du modus ponens. La règle de résolution est également utilisable si C est manquant ou si A et C sont manquants. Dans ce dernier cas, la clause vide peut être dérivée de la contradiction B ¬B (Exercice 2.7 page 30).

2.4 Résolution

Nous généralisons à nouveau la règle de résolution en autorisant les clauses avec un nombre arbitraire de littéraux. Avec les littéraux A1,...,Am, B, C1,...,Cn, la règle générale de résolution s'écrit

$$\frac{(A1 \quad \cdots \quad Am \quad B),\ (\neg B \quad C1 \quad \cdots \quad Cn)}{(A1 \quad \cdots \quad Am \quad C1 \quad \cdots \quad Cn)}. \tag{2.2}$$

Nous appelons les littéraux B et ¬B complémentaires. La règle de résolution supprime une paire de littéraux complémentaires des deux clauses et combine le reste des littéraux dans une nouvelle clause.

Pour prouver qu'à partir d'une base de connaissances KB, une requête Q suit, nous effectuons une preuve par contradiction. En suivant le théorème 2.3 page 20 , nous devons montrer qu'une contradiction peut être dérivée de KB ¬Q. Dans les formules en forme normale conjonctive, une contradiction apparaît sous la forme de deux clauses (A) et (¬A), qui conduisent à la clause vide comme leur résolvante. Le théorème suivant nous assure que ce processus fonctionne vraiment comme souhaité.

Pour que le calcul soit complet, il nous faut un petit ajout, comme le montre l'exemple suivant. Soit la formule (A A) comme base de connaissances. Pour montrer par la règle de résolution qu'à partir de là on peut dériver (A A), il faut montrer que la clause vide peut être dérivée de (A A) (¬A ¬A). Avec la seule règle de résolution, c'est impossible. Avec la factorisation, qui permet de supprimer des copies de littéraux des clauses, ce problème est éliminé. Dans l'exemple, une double application de la factorisation conduit à (A) (¬A), et une étape de résolution à la clause vide.

Théorème 2.5 Le calcul de résolution pour la preuve de l'insatisfaisabilité des formules sous forme normale conjonctive est correct et complet.

Parce que c'est le travail du calcul de résolution de dériver une contradiction de KB ¬Q, il est très important que la base de connaissances KB soit cohérente :

Définition 2.9 Une formule KB est dite cohérente s'il est impossible d'en déduire une contradiction, c'est-à-dire une formule de la forme φ ¬φ.

Sinon, tout peut être dérivé de KB (voir Exercice 2.8 à la page 30). Ce est vrai non seulement pour la résolution, mais aussi pour de nombreux autres calculs.

Parmi les calculs de déduction automatique, la résolution joue un rôle exceptionnel. Nous souhaitons donc travailler un peu plus étroitement avec lui. Contrairement à d'autres calculs, la résolution n'a que deux règles d'inférence et fonctionne avec des formules sous forme normale conjonctive.
Cela rend sa mise en œuvre plus simple. Un autre avantage par rapport à de nombreux calculs réside dans sa réduction du nombre de possibilités d'application des règles d'inférence à chaque étape de la preuve, ce qui réduit l'espace de recherche et diminue le temps de calcul.

A titre d'exemple, nous commençons par un puzzle logique simple qui permet les étapes importantes d'une preuve de résolution à montrer.

Exemple 2.2 Le puzzle logique numéro 7, intitulé Une charmante famille anglaise, tiré du livre allemand [Ber89] se lit (traduit en anglais) :

> Malgré l'étude de l'anglais pendant sept longues années avec un brillant succès, je dois admettre que lorsque j'entends des anglophones parler anglais, je suis totalement perplexe. Récemment, mû par de nobles sentiments, j'ai pris trois auto-stoppeurs, un père, une mère et une fille, dont j'ai vite compris qu'ils étaient anglais et ne parlaient qu'anglais. A chacune des phrases qui suivent, j'hésitais entre deux interprétations possibles. Ils m'ont dit ce qui suit (le deuxième sens possible est entre parenthèses) : Le père : « Nous allons en Espagne (nous sommes de Newcastle) ».
> La mère : "Nous n'allons pas en Espagne et sommes de Newcastle (nous nous sommes arrêtés à Paris et n'allons pas en Espagne)." La fille : "Nous ne sommes pas de Newcastle (nous nous sommes arrêtés à Paris)." Et cette charmante famille anglaise ?

Pour résoudre ce type de problème, nous procédons en trois étapes : formalisation, transformation en forme normale et preuve. Dans de nombreux cas, la formalisation est de loin l'étape la plus difficile car il est facile de faire des erreurs ou d'oublier de petits détails. (Ainsi, l'exercice pratique est très important. Voir Exercices 2.9 à 2.11.)

Ici, nous utilisons les variables S pour "Nous allons en Espagne", N pour "Nous venons de Newcastle », et P pour « Nous nous sommes arrêtés à Paris » et obtiennent comme formalisation les trois propositions de père, mère et fille

$$(S \quad N) \quad [(\neg S \quad N) \quad (P \quad \neg S)] \quad (\neg N \quad P).$$

La factorisation de ¬S dans la sous-formule du milieu amène la formule dans CNF en une seule étape. La numérotation des clauses avec des indices indicés donne

$$KB \equiv (S \quad N)_1 \quad (\neg S)_2 \quad (P \quad N)_3 \quad (\neg N \quad P)_4.$$

Nous commençons maintenant la preuve de résolution, dans un premier temps toujours sans requête Q. Une expression de la forme « Res(m, n) : clausek » signifie que clause est obtenue par résolution de la clause m avec la clause n et est numérotée k.

Rés(1, 2): $(N)_5$

Res(3, 4): $(P)_6$

Res(1, 4): $(S \quad P)_7$

Nous aurions pu également dériver la clause P de Res(4, 5) ou Res(2, 7). Chaque nouvelle étape de résolution conduirait à la dérivation de clauses déjà disponibles. Puisqu'elle ne permet pas la dérivation de la clause vide, il a donc été montré que la base de connaissances est non contradictoire. Jusqu'ici nous avons dérivé N et P. Pour montrer que ¬S est vrai, nous ajoutons la clause $(S)_8$ à l'ensemble des clauses comme une requête négative. Avec l'étape de résolution

Rés(2, 8): $()_9$

la preuve est complète. Ainsi ¬S N P est vérifié. La "charmante famille anglaise" vient évidemment de Newcastle, s'est arrêtée à Paris, mais ne va pas en Espagne.

Exemple 2.3 Le puzzle logique numéro 28 de [Ber89], intitulé The High Jump, se lit

Trois filles pratiquent le saut en hauteur pour leur examen final d'éducation physique. La barre est fixée à 1,20 mètre. « Je parie », dit la première fille à la seconde, « que je m'en sortirai si, et seulement si, tu ne le fais pas ». Si la deuxième fille disait la même chose à la troisième, qui à son tour disait la même chose à la première, serait-il possible que toutes les trois gagnent leur pari ?

Nous montrons par preuve par résolution que tous les trois ne peuvent pas gagner leur pari.
Formalisation:

Le saut de la première fille réussit : A, le saut de la deuxième fille réussit : B, le saut de la troisième fille réussit : C.	Pari de la première fille : (A ¬B), pari de la deuxième fille : (B ¬C), pari de la troisième fille : (C ¬A).

Réclamation : les trois ne peuvent pas tous gagner leur pari :

$$Q \equiv \neg((A \quad \neg B) \quad (B \quad \neg C) \quad (C \quad \neg A))$$

Il faut maintenant montrer par résolution que ¬Q est insatisfaisant.
Transformation en CNF : Premier pari féminin :

$$(\text{UNE} \quad \neg B) \equiv (A \quad \neg B) \quad (\neg B \quad \text{UNE}) \equiv (\neg A \quad \neg B) \quad (A \quad B)$$

Les paris des deux autres filles subissent des transformations analogues, et nous obtenons la créance annulée

$$\neg Q \equiv (\neg A \quad \neg B)1 \quad (A \quad B)2 \quad (\neg B \quad \neg C)3 \quad (B \quad C)4 \quad (\neg C \quad \neg A)5 \quad (C \quad A)6.$$

De là, nous dérivons la clause vide en utilisant la résolution :

Res(1, 6): (C ¬B)7
Rés(4, 7): (C)8
Res(2, 5): (B ¬C)9
Res(3, 9): (¬C)10
Rés(8, 10): ()

Ainsi, l'affirmation a été prouvée.

2.5 Clauses de klaxon

Une clause en forme normale conjonctive contient des littéraux positifs et négatifs et peut être représentée sous la forme

$$(\neg A1 \quad \cdots \quad \neg Am \quad B1 \quad \cdots \quad Bn)$$

avec les variables A1,...,Am et B1,...,Bn. Cette clause peut être transformée en deux étapes simples dans la forme équivalente

$$A1 \quad \cdots \quad Am \quad B1 \quad \cdots \quad Bn.$$

Cette implication contient la prémisse, une conjonction de variables et la conclusion, une disjonction de variables. Par exemple, « s'il fait beau et qu'il y a de la neige au sol, j'irai skier ou je travaillerai ». est une proposition de cette forme. Le destinataire de ce message sait avec certitude que l'expéditeur ne va pas nager.
Une déclaration beaucoup plus claire serait "S'il fait beau et qu'il y a de la neige au sol, j'irai skier". Le récepteur sait maintenant définitivement. Ainsi, nous appelons les clauses avec au plus un littéral positif des clauses définies. Ces clauses ont l'avantage de ne permettre qu'une seule conclusion et sont donc nettement plus simples à interpréter. De nombreuses relations peuvent être décrites par des clauses de ce type. Nous définissons donc

Définition 2.10 Clauses avec au plus un littéral positif de la forme

$$(\neg A1 \quad \cdots \quad \neg Am \quad B) \text{ ou } (\neg A1 \quad \cdots \quad \neg Am) \text{ ou } B$$

ou équivalent)

$$A1 \quad \cdots \quad Am \quad B \text{ ou } A1 \quad \cdots \quad Am \quad f \text{ ou } B.$$

sont nommées clauses de Horn (du nom de leur inventeur). Une clause avec un seul littéral positif est un fait. Dans les clauses avec un littéral négatif et un littéral positif, le littéral positif est appelé la tête.

Pour mieux comprendre la représentation des clauses de Horn, le lecteur peut les déduire des définitions des équivalences que nous utilisons actuellement (Exercice 2.12 page 30).

Les clauses de Horn sont plus faciles à manier non seulement dans la vie quotidienne, mais aussi dans le raisonnement formel, comme on peut le voir dans l'exemple suivant. Supposons que la base de connaissances se compose des clauses suivantes (le " " liant les clauses est omis ici et dans le texte qui suit) :

(beau_temps)1

(chute de neige)2

(chute de neige neige)3

(beau_temps neige ski)4

Si nous voulons maintenant savoir si le ski tient, cela peut facilement être dérivé. Un modus ponens légèrement généralisé suffit ici comme règle d'inférence :

$$\frac{A1 \quad \cdots \quad Am, A1 \quad \cdots \quad Am \quad B}{B}.$$

La preuve de « skier » a la forme suivante (MP(i1,...,ik) représente l'application du modus ponens sur les clauses i1 à ik :

MP(2, 3) : (neige)5 MP(1,
5, 4) : (ski)6.

Avec modus ponens, nous obtenons un calcul complet pour les formules qui consistent en des clauses de Horn de la logique propositionnelle. Dans le cas de grandes bases de connaissances, cependant, le modus ponens peut dériver de nombreuses formules inutiles si l'on commence avec les mauvaises clauses. Par conséquent, dans de nombreux cas, il est préférable d'utiliser un calcul qui commence par la requête et remonte jusqu'à ce que les faits soient atteints. De tels systèmes sont désignés chaînage arrière, contrairement aux systèmes de chaînage avant , qui commencent par des faits et dérivent finalement la requête, comme dans l'exemple ci-dessus avec le modus ponens.

Pour le chaînage arrière des clauses Horn, la résolution SLD est utilisée. SLD est l'abréviation de "Résolution linéaire pilotée par une règle de sélection pour les clauses définies". Dans l'exemple ci-dessus, augmenté de la requête négative (ski f)

(beau_temps)1
(chutes de
neige)2 (chutes de
neige neige)3 (beau_temps neige ski)4
(ski f)5

nous effectuons la résolution SLD en commençant par les étapes de résolution qui découlent de cette clause

Res(5, 4): (beau_temps neige f)6
Res(6, 1): (neige f)7
Res(7, 3): (chute de neige f)8
Rés(8, 2): ()

et déduire une contradiction avec la clause vide. Ici, nous pouvons facilement voir une "résolution linéaire", ce qui signifie que le traitement ultérieur est toujours effectué sur la clause actuellement dérivée. Cela conduit à une grande réduction de l'espace de recherche. De plus, les littéraux de la clause actuelle sont toujours traités dans un ordre fixe (par exemple, de droite à gauche) ("Selection rule driven"). Les littéraux de la clause courante sont appelés

sous-but. Les littéraux de la requête inversée sont les objectifs. La règle d'inférence pour une étape se lit

$$\frac{A1 \quad \cdots \quad Am \qquad B1, B1 \qquad B2 \quad \cdots \quad Bn \qquad f}{A1 \quad \cdots \quad Am \qquad B2 \quad \cdots \quad Bn \qquad f}.$$

Avant l'application de la règle d'inférence, B1, B2,...,Bn — les sous-buts courants — doivent être prouvés. Après l'application, B1 est remplacé par le nouveau sous-objectif A1 ··· Am. Pour montrer que B1 est vrai, nous devons maintenant montrer que A1 ··· Am sont vrais. Ce processus continue jusqu'à ce que la liste des sous-buts des clauses actuelles (la soi-disant pile de buts) soit vide. Avec cela, une contradiction a été trouvée. Si, pour un sous-but ¬Bi , il n'y a pas de clause avec le littéral complémentaire Bi comme tête de clause, la preuve se termine et aucune contradiction ne peut être trouvée. La requête est donc indémontrable.

La résolution SLD joue un rôle important dans la pratique car les programmes du langage de programmation logique PROLOG sont constitués de clauses Horn logiques à prédicats et leur traitement est réalisé au moyen de la résolution SLD (voir Exercice 2.13 page 30 ou Chap . 5).

2.6 Calculabilité et complexité

La méthode de la table de vérité, en tant que système de preuve sémantique le plus simple pour la logique propositionnelle, représente un algorithme qui peut déterminer chaque modèle de n'importe quelle formule en un temps fini. Ainsi, les ensembles de formules insatisfaisables, satisfaisables et valides sont décidables. Le temps de calcul de la méthode de la table de vérité pour la satisfiabilité croît dans le pire des cas de manière exponentielle avec le nombre n de variables car la table de vérité $^{n\,2}$ a des lignes. Une optimisation, la méthode des arbres sémantiques, évite de regarder des variables qui n'apparaissent pas dans les clauses, et économise ainsi du temps de calcul dans de nombreux cas, mais dans le pire des cas, elle est également exponentielle.

En résolution, dans le pire des cas, le nombre de clauses dérivées croît exponentiellement avec le nombre de clauses initiales. Pour départager les deux processus, nous pouvons donc utiliser la règle empirique selon laquelle dans le cas de nombreuses clauses avec peu de variables, la méthode de la table de vérité est préférable, et dans le cas de peu de clauses avec beaucoup de variables, la résolution se terminera probablement plus rapidement. .

La question demeure : la preuve en logique propositionnelle peut-elle aller plus vite ? Existe-t-il de meilleurs algorithmes ? La réponse : probablement pas. Après tout, S. Cook, le fondateur de la théorie de la complexité, a montré que le problème 3-SAT est NP-complet. 3-SAT est l'ensemble de toutes les formules CNF dont les clauses ont exactement trois littéraux. Ainsi il est clair qu'il n'y a probablement (modulo le problème P/NP) aucun algorithme polynomial pour 3-SAT, et donc probablement pas général non plus. Pour les clauses de Horn, cependant, il existe un algorithme dans lequel le temps de calcul pour tester la satisfaisabilité n'augmente que de manière linéaire à mesure que le nombre de littéraux dans la formule augmente.

2.7 Applications et limites

Les démonstrateurs de théorèmes pour la logique propositionnelle font partie de l'ensemble d'outils quotidien du développeur en technologie numérique. Par exemple, la vérification des circuits numériques et la génération de mires de test pour tester les microprocesseurs en fabrication font partie de ces tâches. Des systèmes de preuve spéciaux qui fonctionnent avec des diagrammes de décision binaires (BDD) () sont également utilisés comme structure de données pour le traitement de formules logiques propositionnelles.

En IA, la logique propositionnelle est employée dans des applications simples. Par exemple, des systèmes experts simples peuvent certainement fonctionner avec la logique propositionnelle. Cependant, les variables doivent toutes être discrètes, avec seulement quelques valeurs, et il ne peut y avoir de relations croisées entre les variables. Les connexions logiques complexes peuvent être exprimées de manière beaucoup plus élégante en utilisant la logique des prédicats.

La logique probabiliste est une combinaison très intéressante et actuelle de logique propositionnelle et de calcul probabiliste qui permet de modéliser des connaissances incertaines.
Il est traité en détail au Chap. 7. La logique floue, qui autorise une infinité de valeurs de vérité, est également abordée dans ce chapitre.

2.8 Exercices

Exercice 2.1 Donner une grammaire de la forme Backus–Naur pour la syntaxe des
logique.

Exercice 2.2 Montrer que les formules suivantes sont des tautologies : (a) ¬(A B)
¬A ¬B (b) A B ¬B ¬A (c) ((A
B) (B A)) (A B) (d) (A
B) (¬B C) (A C)

Exercice 2.3 Transformer les formules suivantes en forme normale conjonctive : (a) A B (b) A B A
B (c) A (A
B) B

Exercice 2.4 Vérifiez la satisfaisabilité ou la validité des énoncés suivants. (a) (play_lottery six_right)
gagnant (b) (play_lottery six_right (six_right
win)) win (c) ¬(¬gas_in_tank (gas_in_tank ¬car_starts) ¬car_starts)

Exercice 2.5 À l'aide du langage de programmation de votre choix, programmez un démonstrateur de théorèmes pour la logique propositionnelle en utilisant la méthode des tables de vérité pour les formules sous forme normale conjonctive. Pour éviter une vérification coûteuse de la syntaxe des formules, vous pouvez représenter les clauses sous forme de listes ou d'ensembles de littéraux, et les formules sous forme de listes ou d'ensembles de clauses. Le programme devrait indiquer si la formule est insatisfaisante, satisfaisable ou vraie, et produire le nombre d'interprétations et de modèles différents.

Exercice 2.6 (a)

Montrer que modus ponens est une règle d'inférence valide en montrant que A (A B) | B. (b) Montrer que la
règle de résolution (2.1) est une règle d'inférence valide.

Exercice 2.7 Montrer par application de la règle de résolution que, en normal conjonctif
forme, la clause vide est équivalente à la déclaration fausse.

Exercice 2.8 Montrer que, avec résolution, on peut « dériver » toute clause arbitraire de
une base de connaissances qui contient une contradiction.

Exercice 2.9 Formalisez les fonctions logiques suivantes avec les opérateurs logiques et montrez que votre formule est valide. Présenter le résultat en CNF. (a) L'opération XOR
(ou exclusif) entre deux variables. (b) L'énoncé au moins deux des trois
variables A, B, C sont vraies.

Exercice 2.10 Résolvez le cas suivant à l'aide d'une preuve de résolution : « Si le criminel avait un complice, alors il est venu en voiture. Le criminel n'avait pas de complice et n'avait pas la clé, ou il avait la clé et un complice. Le criminel avait la clé. Le criminel est-il venu en voiture ou pas ? »

Exercice 2.11 Montrer par résolution que la formule de (a) Exercice
2.2(d) est une tautologie. (b) L'exercice
2.4(c) n'est pas satisfaisant.

Exercice 2.12 Démontrer les équivalences suivantes, qui sont importantes pour travailler avec les clauses de Horn : (a)
(¬A1 ··· ¬Am B) ≡ A1 ··· Am B (b) (¬A1 ·
·· ¬Am) ≡ A1 ··· Am f (c) UNE ≡ w UNE

Exercice 2.13 Montrer par résolution SLD que l'ensemble de clauses de Horn suivant est insatisfaisant.

(A)1	(D)4	(A D G)7
(B)2	(E)5	(C F E H)8
(C)3	(A B C F)6	(H f)9

Exercice 2.14 Dans la Sect. 2.6, il est écrit : "Ainsi, il est clair qu'il n'y a probablement (modulo le problème P/NP) aucun algorithme polynomial pour 3-SAT, et donc probablement pas général non plus." Justifiez le "probablement" dans cette phrase.

Logique des prédicats du premier ordre 3

De nombreux problèmes pratiques et pertinents ne peuvent pas être ou ne peuvent être formulés que très incommodément dans le langage de la logique propositionnelle, comme nous pouvons facilement le reconnaître dans l'exemple suivant. La déclaration

"Robot 7 est situé à la position xy (35, 79)"

peut en fait être directement utilisée comme variable logique propositionnelle

"Robot_7_is_situated_at_xy_position_(35, 79)"

pour raisonner avec la logique propositionnelle, mais raisonner avec ce genre de proposition est très gênant. Supposons que 100 de ces robots puissent s'arrêter n'importe où sur une grille de 100 × 100 points. Pour décrire chaque position de chaque robot, nous aurions besoin de 100 · 100 · 100 = 1 000 000 = 106 variables différentes. La définition des relations entre les objets (ici les robots) devient vraiment difficile. La relation

"Le robot A est à droite du robot B."

n'est sémantiquement rien de plus qu'un ensemble de paires. Sur les 10 000 paires possibles d'abscisses, il y a (99 · 98)/2 = 4851 paires ordonnées. Avec les 10 000 combinaisons de valeurs y possibles pour les deux robots, il y a (100 · 99) = 9900 pour les mulas du type

Robot_7_is_to_the_right_of_robot_12
Robot_7_is_situated_at_xy_position_(35, 79)
Robot_12_is_situated_at_xy_position_(10, 93) ···

définissant ces relations, chacune d'elles avec $\binom{104}{2} \cdot 0{,}485 = 0{,}485 \cdot 108$ variantes côté droit. Dans la logique des prédicats du premier ordre, on peut définir pour cela un prédicat Position(number, xPosition, yPosition). La relation ci-dessus ne doit plus être énumérée comme un grand nombre de paires, mais elle est plutôt décrite abstraitement avec une règle de la forme

$$\forall u \; \forall v \; \text{is\_further\_right}(u, v) \Leftrightarrow \exists xu \; \exists yu \; \exists xv \; \exists yv \; \text{position}(u, xu, yu) \wedge \text{position}(v, xv, yv) \wedge xu > xv,$$

Où $\forall$ u se lit comme « pour tout u » et $\exists$ v comme « il existe v ».

W. Ertel, Introduction à l'intelligence artificielle,
Sujets de premier cycle en informatique, DOI
10.1007/978-0-85729-299-5_3,

Dans ce chapitre, nous allons définir la syntaxe et la sémantique de la logique des prédicats du premier ordre (PL1), montrer que de nombreuses applications peuvent être modélisées avec ce langage, et montrer qu'il existe un calcul complet et sain pour ce langage.

3.1 Syntaxe

Premièrement, nous solidifions la structure syntaxique des termes.

Définition 3.1 Soient V un ensemble de variables, K un ensemble de constantes et F un ensemble de symboles de fonctions. Les ensembles VK et F sont deux à deux disjoints. Nous définissons l'ensemble des termes de manière récursive : •
Toutes les variables et constantes sont des termes (atomiques). •
Si t1,...,tn sont des termes et f un symbole de fonction à n positions, alors f (t1,...,tn) est aussi un terme.

Quelques exemples de termes sont f (sin(ln(3)), exp(x)) et g(g(g(x))). Pouvoir établir des relations logiques entre les termes, nous construisons des formules à partir des termes.

Définition 3.2 Soit P un ensemble de symboles de prédicat. Les formules logiques de prédicat sont construites comme suit : •
Si t1,...,tn sont des termes et p un symbole de prédicat à n places, alors p(t1,...,tn) est une formule (atomique). • Si A et B sont des
formules, alors ¬A, (A), A B, A B, A B, A B sont aussi des formules.

• Si x est une variable et A une formule, alors x A et x A sont aussi des formules.
est le quantificateur universel et le quantificateur existentiel.
• p(t1,...,tn) et ¬p(t1,...,tn) sont appelés littéraux. • Les formules dans
lesquelles chaque variable est dans la portée d'un quantificateur sont appelées phrases du premier ordre ou formules fermées. Les variables qui ne sont pas dans la portée d'un quantificateur sont appelées variables libres. • Les définitions 2.8 (CNF) et 2.10 (clauses
de Horn) sont valables pour les formules de littéraux de la logique des prédicats de manière analogue.

Dans le tableau 3.1 à la page 33 , plusieurs exemples de formules PL1 sont donnés avec avec leurs interprétations intuitives.

Tableau 3.1 Exemples de formules en logique de prédicat du premier ordre. S'il vous plaît noter que la mère ici est un symbole de fonction

Formule	Description
x grenouille(x) verte(x)	Toutes les grenouilles sont vertes
x grenouille(x) marron(x) grosse(x)	Toutes les grenouilles brunes sont grosses
x aime(x, gâteau)	Tout le monde aime le gâteau
¬ x aime(x, gâteau)	Tout le monde n'aime pas les gâteaux
¬ x aime(x, gâteau)	Personne n'aime le gâteau
x y aime(y, x)	Il y a quelque chose que tout le monde aime
x y aime(x, y)	Il y a quelqu'un qui aime tout
x y aime(y, x)	Tout est aimé par quelqu'un
x y aime(x, y)	Tout le monde aime quelque chose
x client(x) aime(bob ,x) x client(x)	Bob aime chaque client
aime(x, bob) x boulanger(x)	Il y a un client que Bob aime
y client(y) mag(x, y)	Il y a un boulanger qui aime tous ses clients
x plus âgée(mère(x), x) x	Chaque mère est plus âgée que son enfant
plus âgée(mère(mère(x)), x)	Chaque grand-mère est plus âgée qu'elle l'enfant de la fille
x y z rel(x, y) rel(y, z) rel(x, z)	rel est une relation transitive

3.2 Sémantique

Dans la logique propositionnelle, chaque variable se voit directement attribuer une valeur de vérité par une interprétation. Dans la logique des prédicats, la signification des formules est définie de manière récursive sur construction de la formule, en ce que nous attribuons d'abord des constantes, des variables et une fonction symboles aux objets du monde réel.

Définition 3.3 Une interprétation I est définie comme

- Une application de l'ensemble des constantes et des variables K V à un ensemble W de noms d'objets dans le monde.
- Une correspondance entre l'ensemble des symboles de fonction et l'ensemble des fonctions dans le monde. Chaque symbole de fonction à n places se voit attribuer une fonction à n places.
- Une correspondance entre l'ensemble des symboles de prédicat et l'ensemble des relations dans le monde. Chaque symbole de prédicat à n places se voit attribuer une relation à n places.

Exemple 3.1 Soit c1, c2, c3 des constantes, « plus » un symbole de fonction à deux chiffres, et "gr" un symbole de prédicat à deux places. La vérité de la formule

$$F \equiv gr(plus(c1, c3), c2)$$

dépend de l'interprétation I. Nous choisissons d'abord l'interprétation évidente suivante des constantes, de la fonction et des prédicats dans les nombres naturels :

$$I1 : c1 \to 1, c2 \to 2, c3 \to 3, plus \to +, gr \to >$$

Ainsi, la formule est mappée à

$$1 + 3 > 2, \text{ ou après évaluation } 4 > 2.$$

La relation supérieure à sur l'ensemble {1, 2, 3, 4} est l'ensemble de toutes les paires (x, y) de nombres avec x> y, c'est-à-dire l'ensemble G = {(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)}. Comme (4, 2) G, la formule F est vraie sous l'interprétation I1. Cependant, si nous choisissons l'interprétation

$$I2 : c1 \to 2, c2 \to 3, c3 \to 1, plus \to -, gr \to >,$$

on obtient

$$2 - 1 > 3, \text{ ou } 1 > 3.$$

Le couple (1, 3) n'est pas membre de G. La formule F est fausse sous l'interprétation I2. Évidemment, la vérité d'une formule en PL1 dépend de l'interprétation. Maintenant, après cet aperçu, nous définissons la vérité.

Définition 3.4 •

Une formule atomique p(t1,...,tn) est vraie (ou valide) sous l'interprétation I si, après interprétation et évaluation de tous les termes t1,...,tn et interprétation du prédicat p par la relation à n places r , il soutient que

$$(I(t1), \ldots, I(tn)) \quad r.$$

- La vérité des formules sans quantificateur découle de la vérité des formules atomiques — comme dans le calcul propositionnel — à travers la sémantique des opérateurs logiques définis dans le tableau 2.1

page 17. • Une formule x F est vraie sous l'interprétation I exactement quand elle est vrai étant donné un changement arbitraire de l'interprétation pour la variable x (et uniquement pour x)

- Une formule x F est vraie sous l'interprétation I exactement quand il existe une interprétation pour x qui rend la formule vraie.

Les définitions de l'équivalence sémantique des formules, pour les concepts satisfaisant, vrai, insatisfiable et modèle, ainsi que l'implication sémantique (Définitions 2.4, 2.5, 2.6) sont transférées telles quelles du calcul propositionnel à la logique des prédicats.

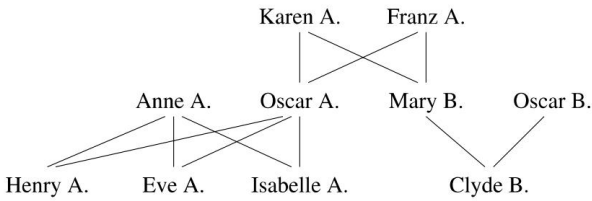


Fig. 3.1 Un arbre généalogique. Les arêtes allant de Clyde B. vers le haut jusqu'à Mary B. et Oscar B. représentent l'élément (Clyde B., Mary B., Oscar B.) en tant que relation d'enfant

Théorème 3.1 Les théorèmes 2.2 (théorème de déduction) et 2.3 (preuve par contradiction) sont valables de manière analogue pour PL1.

Exemple 3.2 L'arbre généalogique de la Fig. 3.1 représente graphiquement (au niveau sémantique) la relation

Enfant = {(Oscar A., Karen A., Frank A.), (Mary B., Karen A., Frank A.), (Henry A., Anne A., Oscar A.), (Eve A., Anne A., Oscar A.), (Isabelle A., Anne A., Oscar A.), (Clyde B., Mary B., Oscar B.)}

Par exemple, le triplet (Oscar A., Karen A., Frank A.) représente la proposition « Oscar A. est un enfant de Karen A. et Frank A. ». D'après les noms, nous lisons la relation à une place

Femme = {Karen A., Anne A., Mary B., Eve A., Isabelle A.}

des femmes. Nous voulons maintenant établir des formules pour les relations familiales. Nous définissons d'abord un prédicat à trois places enfant(x, y, z) avec la sémantique

$$I(\text{enfant}(x, y, z)) = w \equiv (I(x), I(y), I(z)) \quad \text{Genre.}$$

Sous l'interprétation I(oscar) = Oscar A., I(eve) = Eve A., I(anne) = Anne A., il est également vrai que child(eve, anne, oscar). Pour que enfant(eve, oscar, anne) soit vrai, il faut, avec

$$x \quad y \quad z \ \text{enfant}(x, y, z) \quad \text{enfant}(x, z, y),$$

symétrie du prédicat enfant dans les deux derniers arguments. Pour plus de définitions, nous nous référons à l'exercice 3.1 à la page 54 et définissons le descendant du prédicat de manière récursive

comme

$$x \quad y \ \text{descendant}(x, y) \quad z \ \text{enfant}(x, y, z) \quad (\ u \quad v \ \text{enfant}(x, u, v) \quad \text{descendant}(u, y)).$$

Maintenant, nous construisons une petite base de connaissances avec des règles et des faits. Laisser

KB ≡ femelle(karen) femelle(anne) femelle(mary)
femelle(eve) femelle(isabelle)
enfant (oscar, karen, franz) enfant (marie, karen, franz)
enfant(eve, anne, oscar) enfant(henry, anne, oscar)
enfant(isabelle, anne, oscar) enfant(clyde,mary, oscarb)
(x y z enfant(x, y, z) enfant(x, z, y))
(x y descendant(x, y) z enfant(x, y, z)
(u v enfant(x, u, v) descendant(u, y))).

On peut maintenant se demander, par exemple, si les propositions enfant(eve, oscar, anne) ou descendant(eve,franz) sont dérivables. Pour cela, nous avons besoin d'un calcul.

3.2.1 Égalité

Pour pouvoir comparer des termes, l'égalité est une relation très importante dans la logique des prédicats. L'égalité des termes en mathématiques est une relation d'équivalence, c'est-à-dire qu'elle est réflexive, symétrique et transitive. Si nous voulons utiliser l'égalité dans les formules, nous devons soit incorporer ces trois attributs sous forme d'axiomes dans notre base de connaissances, soit intégrer l'égalité dans le calcul. Nous prenons la voie facile et définissons un prédicat « = » qui, s'écartant de la Définition 3.2 à la page 32, est écrit en utilisant la notation infixe comme il est d'usage en mathématiques. (Une équation x = y pourrait bien sûr aussi s'écrire sous la forme eq(x, y).) Ainsi, les axiomes d'égalité ont la forme

$$\begin{array}{lll} x\,x = x & & \text{(réflexivité)} \\ x\,y\,x = y \quad y = x & & \text{(symétrie)} \\ x\,y\,z\,x = y \quad y = z \quad x = z & & \text{(transitivité).} \end{array} \tag{3.1}$$

Pour garantir l'unicité des fonctions, nous exigeons en outre

$$x\,y\,x = y \quad f(x) = f(y) \text{ (axiome de substitution)} \tag{3.2}$$

pour chaque symbole de fonction. De manière analogue, nous exigeons pour tous les symboles de prédicat

$$x\,y\,x = y \quad p(x) \quad p(y) \text{ (axiome de substitution).} \tag{3.3}$$

Nous formulons d'autres relations mathématiques, comme la relation « < », par des moyens similaires (Exercice 3.4 à la page 55).

Souvent une variable doit être remplacée par un terme. Pour le faire correctement et décrire simplement, nous donnons la définition suivante.

Définition 3.5 On note [x/t] la formule qui résulte du remplacement de chaque occurrence libre de la variable x dans par le terme t. Ainsi, nous n'autorisons aucune variable dans le terme t qui soit quantifiée en . Dans ces cas, les variables doivent être renommées pour garantir cela.

Exemple 3.3 Si, dans la formule xx = y, la variable libre y est remplacée par le terme x + 1, le résultat est xx = x + 1. Avec une substitution correcte, nous obtenons la formule xx = y + 1, qui a une sémantique très différente.

3.3 Quantificateurs et formes normales

D'après la Définition 3.4 page 34, la formule xp(x) est vraie si et seulement si elle est vraie pour toutes les interprétations de la variable x. Au lieu du quantificateur, on pourrait écrire p(a1) ··· p(an) pour toute constante a1 ··· an dans K. Pour xp(x) on pourrait écrire p(a1) ··· poêle). Il en résulte avec la loi de Morgan que

x ≡ ¬ x¬ .

Par cette équivalence, les quantificateurs universels et existentiels sont mutuellement remplaçables.

Exemple 3.4 La proposition « Tout le monde veut être aimé » est équivalente à la proposition « Personne ne veut être aimé ».

Les quantificateurs sont une composante importante du pouvoir expressif de la logique des prédicats. Cependant, ils perturbent l'inférence automatique en IA car ils complexifient la structure des formules et augmentent le nombre de règles d'inférence applicables à chaque étape d'une preuve. Par conséquent, notre prochain objectif est de trouver, pour chaque formule logique de prédicat, une formule équivalente sous une forme normale standardisée avec le moins de quantificateurs possible. Dans un premier temps, nous amenons les quantificateurs universels au début de la formule et définissons ainsi

Définition 3.6 Une formule logique de prédicat est sous forme normale prénexe si elle admet que • =

Q1x1 ··· Qnxn ψ. • ψ est une formule sans quantificateur. • Qi { , } pour i = 1,...,n.

Undergraduate Topics in Computer Science

Wolfgang Ertel

Introduction to Artificial Intelligence

UTiCS

Springer

Sujets de premier cycle en informatique

Les sujets de premier cycle en informatique (UTiCS) offrent un contenu pédagogique de haute qualité aux étudiants de premier cycle qui étudient dans tous les domaines de l'informatique et des sciences de l'information. Du matériel fondamental et théorique de base aux sujets et applications de dernière année, les livres UTiCS adoptent une approche nouvelle, concise et moderne et sont idéaux pour l'auto-apprentissage ou pour un cours d'un ou deux semestres. Les textes sont tous rédigés par des experts reconnus dans leurs domaines, revus par un comité consultatif international, et contiennent de nombreux exemples et problèmes. Beaucoup incluent des solutions entièrement travaillées.

Pour d'autres
volumes : www.springer.com/series/7592

Wolfgang Ertel

Introduction à l'artificiel Intelligence

Traduit par Nathanaël Black
Avec des illustrations de Florian Mast

Springer

Prof. Dr. Wolfgang Ertel FB
Elektrotechnik und Informatik Hochschule
Ravensburg-Weingarten Université des
sciences appliquées Weingarten
Allemagne
ertel@hs-
weingarten.de

Éditeur de la série
Ian Macky

ISSN 1863-7310
ISBN 978-0-85729-298-8 DOI e-ISBN 978-0-85729-299-5
10.1007/978-0-85729-299-5 Springer
Londres Dordrecht Heidelberg New York

Données de catalogage avant publication de la British Library
Une notice de catalogue pour ce livre est disponible à la British Library

Numéro de contrôle de la Bibliothèque du Congrès : 2011923216

Publié à l'origine en allemand par Vieweg+Teubner, 65189 Wiesbaden, Allemagne, sous le titre « Ertel, Wolfgang : Grundkurs Künstliche Intelligenz. 2. rév. édition". © Vieweg+Teubner | GWV Fachverlage GmbH,
Wiesbaden 2009

Imprimé sur du papier sans acide

Springer fait partie de Springer Science+Business Media (www.springer.com)

Préface

L'intelligence artificielle (IA) a pour objectif précis de comprendre l'intelligence et de construire des systèmes intelligents. Cependant, les méthodes et les formalismes utilisés pour atteindre cet objectif ne sont pas fermement établis, ce qui a abouti à une IA composée aujourd'hui d'une multitude de sous-disciplines. La difficulté d'un cours d'initiation à l'IA réside dans le fait de transmettre un maximum de branches sans perdre trop de profondeur et de précision.

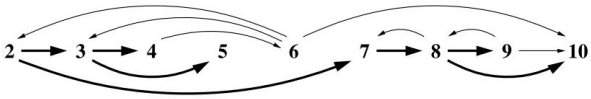
Le livre de Russell et Norvig [RN10] est plus ou moins l'introduction standard à l'IA. Cependant, comme ce livre compte 1 152 pages et qu'il est trop long et trop coûteux pour la plupart des étudiants, les exigences pour l'écriture de ce livre étaient claires : il devrait être une introduction accessible à l'IA moderne pour l'auto-apprentissage ou comme base d'un quatre cours d'une heure, de 300 pages maximum. Le résultat est devant vous.

En l'espace de 300 pages, un domaine aussi vaste que l'IA ne peut être entièrement couvert. Pour éviter de transformer le livre en table des matières, j'ai tenté d'approfondir et d'introduire des algorithmes et des applications concrètes dans chacune des branches suivantes : agents, logique, recherche, raisonnement avec incertitude, apprentissage automatique et réseaux de neurones.

Les domaines du traitement d'images, de la logique floue et du traitement du langage naturel ne sont pas abordés en détail. Le domaine du traitement d'images, qui est important pour toute l'informatique, est une discipline à part entière avec de très bons manuels, comme [GW08]. Le traitement du langage naturel a un statut similaire. Pour reconnaître et générer du texte et du langage parlé, des méthodes issues de la logique, du raisonnement probabiliste et des réseaux de neurones sont appliquées. En ce sens, ce domaine fait partie de l'IA. D'autre part, la linguistique informatique est sa propre branche étendue de l'informatique et a beaucoup en commun avec les langages formels. Dans ce livre, nous indiquerons ces systèmes appropriés à plusieurs endroits, mais nous ne donnerons pas une introduction systématique. Pour une première introduction dans ce domaine, nous renvoyons aux Chap. 22 et 23 dans [RN10]. La logique floue, ou théorie des ensembles flous, est devenue une branche de la théorie du contrôle en raison de son application principale dans la technologie de l'automatisation et est couverte dans les livres et conférences corresp Nous nous passerons donc ici d'une introduction.

Les dépendances entre les chapitres du livre sont grossièrement esquissées dans le graphique ci-dessous. Pour faire simple, Chap. 1, avec l'introduction fondamentale pour tous les chapitres suivants, est omise. À titre d'exemple, la flèche plus épaisse de 2 à 3 signifie que la logique propositionnelle est une condition préalable à la compréhension de la logique des prédicats. La flèche fine de 9 à 10 signifie que les réseaux de neurones sont utiles pour comprendre l'apprentissage par renforcement, mais pas absolument nécessaires. Mince vers l'arrière

les flèches doivent indiquer clairement que les chapitres ultérieurs peuvent donner une compréhension plus approfondie à des sujets déjà appris.



Ce livre est applicable aux étudiants en informatique et autres techniques naturelles sciences et, pour la plupart, exige des connaissances de niveau secondaire en mathématiques. En plusieurs endroits, des connaissances issues de l'algèbre linéaire et de l'analyse multidimensionnelle est nécessaire. Pour une compréhension plus approfondie du contenu, un travail actif sur les exercices est indispensable. Cela signifie que les solutions ne doivent être consultées que après un travail intensif sur chaque problème, et uniquement pour vérifier ses solutions, fidèles à La devise de Léonard de Vinci « Étudier sans dévotion endommage le cerveau ». Quelque peu les problèmes les plus difficiles sont marqués d'un , et les plus difficiles d' un . Les problèmes qui nécessitent de la programmation ou des connaissances particulières en informatique sont étiqueté avec .

Sur le site Web du livre à l'adresse www.hs-weingarten.de/~ertel/aibook , des supports numériques pour les exercices tels que des données d'entraînement pour l'apprentissage d'algorithmes, une page avec des références aux programmes d'IA mentionnés dans le livre, une liste de liens vers les sujets abordés, un liste cliquable de la bibliographie, une liste d'errata et des diapositives de présentation pour les conférenciers peut être trouvé. Je demande au lecteur de bien vouloir envoyer des suggestions, des critiques et des conseils sur erreurs directement à ertel@hs-weingarten.de.

Ce livre est une traduction mise à jour de mon livre allemand "Grundkurs Künstliche Intelligence » publié par Vieweg Verlag. Mes remerciements particuliers vont au traducteur Nathan Black qui, dans une excellente coopération transatlantique entre l'Allemagne et La Californie via SVN, Skype et Email a produit ce texte. Je suis reconnaissant à Franz Kur feß, qui m'a présenté Nathan ; à Matthew Wight pour la relecture de la traduction livre et à Simon Rees de Springer Verlag pour sa patience.

Je tiens à remercier ma femme Evelyn pour son soutien et sa patience durant cette projet chronophage. Des remerciements particuliers vont à Wolfgang Bibel et Chris Loben schuss, qui ont soigneusement corrigé le manuscrit allemand. Leurs suggestions et discussions conduisent à de nombreuses améliorations et ajouts. Pour lire les corrections et autres précieux services, je tiens à remercier Richard Cubek, Celal Döven, Joachim Feßler, Nico Hochgeschwender, Paul Kirner, Wilfried Meister, Norbert Perk, Peter Radtke, Markus Schneider, Manfred Schramm, Uli Stärk, Michel Tokic, Arne Usadel et tous les étudiants intéressés. Mes remerciements vont également à Florian Mast pour la des dessins animés inestimables et une collaboration très efficace.

J'espère que durant vos études ce livre vous aidera à partager ma fascination pour Intelligence artificielle.

Ravensbourg Wolfgang Ertel
Février 2011

Contenu

Introduction 1

1.1 Qu'est-ce que l'intelligence artificielle ?

Le terme intelligence artificielle suscite des émotions. D'une part, il y a notre fascination pour l'intelligence, qui semble donner à nous, humains, une place spéciale parmi les formes de vie. Des questions se posent telles que "Qu'est-ce que l'intelligence?", "Comment peut-on mesurer l'intelligence?" ou "Comment fonctionne le cerveau?". Toutes ces questions sont significatives lorsqu'on essaie de comprendre l'intelligence artificielle. Cependant, la question centrale pour l'ingénieur, en particulier pour l'informaticien, est la question de la machine intelligente qui se comporte comme une personne, montrant un comportement intelligent.

L'attribut artificiel peut éveiller des associations bien différentes. Cela fait craindre des cyborgs intelligents. Il rappelle des images de romans de science-fiction. Cela soulève la question de savoir si notre bien le plus élevé, l'âme, est quelque chose que nous devrions essayer de comprendre, de modéliser ou même de reconstruire.

Avec des interprétations désinvoltes aussi différentes, il devient difficile de définir le terme intelligence artificielle ou IA de manière simple et robuste. Néanmoins, je voudrais essayer, à l'aide d'exemples et de définitions historiques, de caractériser le domaine de l'IA. En 1955, John McCarthy, l'un des pionniers de l'IA, a été le premier à définir le terme intelligence artificielle, à peu près comme suit :

> Le but de l'IA est de développer des machines qui se comportent comme si elles étaient intelligentes.

Pour tester cette définition, le lecteur peut imaginer le scénario suivant. Une quinzaine de petits véhicules robotiques se déplacent sur une surface carrée fermée de quatre mètres sur quatre. On peut observer divers modèles de comportement. Certains véhicules forment de petits groupes avec relativement peu de mouvement. D'autres se déplacent paisiblement dans l'espace et évitent gracieusement toute collision. D'autres encore semblent suivre un leader. Des comportements agressifs sont également observables. Est-ce que nous assistons à un comportement intelligent ?

Selon la définition de McCarthy, les robots susmentionnés peuvent être décrits comme intelligents. Le psychologue Valentin Braitenberg a montré que ce comportement apparemment complexe peut être produit par des circuits électriques très simples [Bra84]. Les véhicules dits Braitenberg ont deux roues, chacune étant entraînée par un moteur électrique indépendant. La vitesse de chaque moteur est influencée par un capteur de lumière sur

W. Ertel, Introduction à l'intelligence artificielle, Sujets de premier cycle en informatique, DOI 10.1007/978-0-85729-299-5_1,

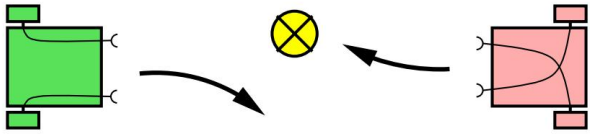


Fig. 1.1 Deux véhicules Braitenberg très simples et leurs réactions à une source lumineuse

l'avant du véhicule comme illustré à la Fig. 1.1. Plus il y a de lumière qui frappe le capteur, plus le moteur tourne vite. Le véhicule 1 sur la partie gauche de la figure, selon sa configuration, s'éloigne d'une source lumineuse ponctuelle. Le véhicule 2, quant à lui, se dirige vers la source lumineuse. D'autres petites modifications peuvent créer d'autres modèles de comportement, de sorte qu'avec ces véhicules très simples, nous pouvons réaliser le comportement impressionnant décrit ci-dessus.

Il est clair que la définition ci-dessus est insuffisante car l'IA a pour objectif de résoudre des problèmes pratiques difficiles qui sont sûrement trop exigeants pour le véhicule Braitenberg. Dans l'Encyclopedia Britannica [Bri91] on trouve une Définition qui ressemble à :

> L'IA est la capacité des ordinateurs numériques ou des robots contrôlés par ordinateur à résoudre des problèmes qui sont normalement associés aux capacités de traitement intellectuel supérieures des humains . . .

Mais cette définition a aussi des faiblesses. Il admettrait par exemple qu'un ordinateur doté d'une grande mémoire, capable de sauvegarder un texte long et de le récupérer à la demande, affiche des capacités intelligentes, car la mémorisation de textes longs peut certainement être considérée comme une capacité de traitement intellectuel supérieure de l'homme, comme peut le faire par exemple la rapidité multiplication de deux nombres à 20 chiffres. Selon cette définition, chaque ordinateur est donc un système d'IA. Ce dilemme est élégamment résolu par la définition suivante d'Elaine Rich [Ric83] :

> L'intelligence artificielle est l'étude de la façon de faire faire aux ordinateurs des choses dans lesquelles, pour le moment, les gens sont meilleurs.

Riche, laconique et concis, caractérise ce que les chercheurs en IA ont fait au cours des 50 dernières années. Même en l'an 2050, cette définition sera à jour.

Des tâches telles que l'exécution de nombreux calculs en peu de temps sont les points forts des ordinateurs numériques. À cet égard, ils surpassent les humains de plusieurs multiples. Dans de nombreux autres domaines, cependant, les humains sont de loin supérieurs aux machines. Par exemple, une personne entrant dans une pièce inconnue reconnaîtra l'environnement en quelques fractions de seconde et, si nécessaire, prendra tout aussi rapidement des décisions et planifiera des actions. À ce jour, cette tâche est trop exigeante pour les robots autonomes1. Selon la définition de Rich, il s'agit donc d'une tâche pour l'IA. En fait, la recherche sur les robots autonomes est un thème important et actuel en IA. La construction d'ordinateurs d'échecs, d'autre part, a perdu de sa pertinence car ils jouent déjà au niveau ou au-dessus du niveau des grands maîtres.

Il serait cependant dangereux de conclure de la définition de Rich que l'IA ne s'intéresse qu'à la mise en œuvre pragmatique de processus intelligents. Les systèmes intelligents, au sens de la définition de Rich, ne peuvent être construits sans une profonde

1Un robot autonome fonctionne de manière autonome, sans assistance manuelle, notamment sans télécommande contrôle.

compréhension du raisonnement humain et de l'action intelligente en général, en raison de laquelle les neurosciences (voir section 1.1.1) sont d'une grande importance pour l'IA. Cela montre également que les autres définitions citées reflètent des aspects importants de l'IA.

Une force particulière de l'intelligence humaine est l'adaptabilité. Nous sommes capables de nous adapter à diverses conditions environnementales et de modifier notre comportement en conséquence grâce à l'apprentissage. Précisément parce que notre capacité d'apprentissage est tellement supérieure à celle des ordinateurs, l'apprentissage automatique est, selon la définition de Rich, un sous-domaine central de l'IA.

1.1.1 Science du cerveau et résolution de problèmes

Grâce à la recherche de systèmes intelligents, nous pouvons essayer de comprendre comment fonctionne le cerveau humain, puis le modéliser ou le simuler sur ordinateur. De nombreuses idées et principes dans le domaine des réseaux de neurones (voir chapitre 9) proviennent de la science du cerveau avec le domaine connexe des neurosciences.

Une approche très différente résulte de l'adoption d'une ligne d'action orientée vers un objectif, en partant d'un problème et en essayant de trouver la solution la plus optimale. La façon dont les humains résolvent le problème est traitée comme sans importance ici. La méthode, dans cette approche, est secondaire. D'abord et avant tout, la solution intelligente optimale au problème. Plutôt que d'employer une méthode fixe (comme, par exemple, la logique des prédicats), l'IA a pour objectif constant la création d'agents intelligents pour autant de tâches différentes que possible. Comme les tâches peuvent être très différentes, il n'est pas surprenant que les méthodes actuellement employées en IA soient souvent aussi très différentes. Semblable à la médecine, qui englobe de nombreuses procédures de diagnostic et de thérapie différentes, souvent vitales, l'IA offre également une large palette de solutions efficaces pour des applications très variées.
Pour l'inspiration mentale, considérez la Fig. 1.2 à la page 4. Tout comme en médecine, il n'existe pas de méthode universelle pour tous les domaines d'application de l'IA, mais plutôt un grand nombre de solutions possibles pour le grand nombre de divers problèmes quotidiens, petits et grands.

Les sciences cognitives sont consacrées à la recherche sur la pensée humaine à un niveau un peu plus élevé. À l'instar de la science du cerveau, ce domaine fournit à l'IA pratique de nombreuses idées importantes. D'autre part, les algorithmes et les implémentations conduisent à d'autres conclusions importantes sur le fonctionnement du raisonnement humain. Ces trois domaines bénéficient ainsi d'un échange interdisciplinaire fructueux. Le sujet de ce livre, cependant, est principalement l'IA axée sur les problèmes en tant que sous-discipline de l'informatique.

Il existe de nombreuses questions philosophiques intéressantes autour de l'intelligence et de l'intelligence artificielle. Nous, les humains, avons une conscience; c'est-à-dire que nous pouvons penser à nous-mêmes et même penser que nous sommes capables de penser à nous-mêmes. Comment se fait la conscience ? De nombreux philosophes et neurologues croient maintenant que l'esprit et la conscience sont liés à la matière, c'est-à-dire au cerveau. La question de savoir si les machines pourraient un jour avoir un esprit ou une conscience pourrait à un moment donné dans le futur devenir pertinente. Le problème corps-esprit concerne en particulier si oui ou non l'esprit est lié au corps. Nous n'aborderons pas ces questions ici. Le lecteur intéressé peut consulter [Spe98, Spe97] et est invité, dans le cadre d'études sur les technologies de l'IA, à se forger une opinion personnelle sur ces questions.

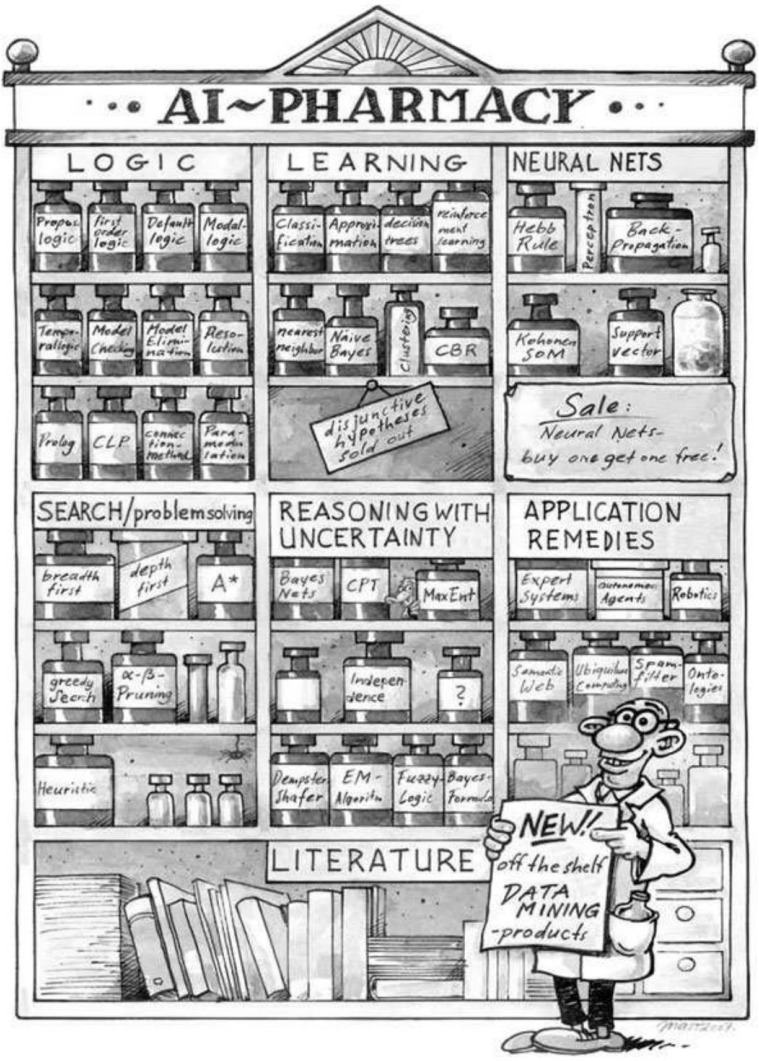


Fig. 1.2 Un petit échantillon des solutions proposées par l'IA

1.1.2 Le test de Turing et les Chatterbots

Alan Turing s'est fait un nom en tant que pionnier de l'IA avec sa définition d'une machine intelligente, dans laquelle la machine en question doit passer le test suivant. La personne testée Alice est assise dans une pièce fermée à clé avec deux terminaux informatiques. Un terminal est connecté à une machine, l'autre à une personne non malveillante Bob. Alice peut taper des questions dans les deux terminaux. Elle est chargée de décider, au bout de cinq minutes, quel terminal appartient à la machine. La machine réussit le test si elle peut tromper Alice au moins 30% du temps [Tur50].

Si le test est très intéressant philosophiquement, pour l'IA pratique, qui traite de la résolution de problèmes, ce n'est pas un test très pertinent. Les raisons à cela sont similaires à celles mentionnées ci-dessus concernant les véhicules Braitenberg (voir exercice 1.3 à la page 14).

Le pionnier de l'IA et critique social Joseph Weizenbaum a développé un programme nommé Eliza, qui est censé répondre aux questions d'un sujet de test comme un psychologue humain [Wei66]. Il était en effet capable de démontrer le succès dans de nombreux cas. Soi-disant, sa secrétaire avait souvent de longues discussions avec le programme. Aujourd'hui, sur Internet, il existe de nombreux soi-disant chatterbots, dont certaines des premières réponses sont assez impressionnantes. Au bout d'un certain temps, cependant, leur caractère artificiel devient apparent. Certains de ces programmes sont réellement capables d'apprendre, tandis que d'autres possèdent des connaissances extraordinaires sur divers sujets, par exemple la géographie ou le développement de logiciels. Il existe déjà des applications commerciales pour les chatterbots dans le support client en ligne et il pourrait y en avoir d'autres dans le domaine de l'e-learning. Il est concevable que l'apprenant et le système e-learning puissent communiquer via un chatterbot.
Le lecteur voudra peut-être comparer plusieurs chatterbots et évaluer leur intelligence dans l'exercice 1.1 à la page 14.

1.2 L'histoire de l'IA

L'IA s'appuie sur de nombreuses réalisations scientifiques passées qui ne sont pas mentionnées ici, car l'IA en tant que science à part entière n'existe que depuis le milieu du XXe siècle. Le tableau 1.1 à la page 10, avec les jalons les plus importants de l'IA, et une représentation graphique des principaux mouvements de l'IA à la Fig. 1.3 à la page 6 complètent le texte suivant.

1.2.1 Les premiers débuts

Dans les années 1930, Kurt Gödel, Alonso Church et Alan Turing ont jeté des bases importantes pour la logique et l'informatique théorique. Les théorèmes de Gödel sont particulièrement intéressants pour l'IA. Le théorème de complétude stipule que la logique des prédicats du premier ordre est complète. Cela signifie que chaque énoncé vrai qui peut être formulé dans la logique des prédicats est démontrable en utilisant les règles d'un calcul formel. Sur cette base, des démonstrateurs automatiques de théorèmes pourraient plus tard être construits comme des implémentations de calculs formels. Avec le théorème d'incomplétude, Gödel a montré que dans les logiques d'ordre supérieur, il existe des énoncés vrais qui ne sont pas démontrables2. Avec cela, il a découvert les limites douloureuses des systèmes formels.

La preuve d'Alan Turing de l'indécidabilité du problème d'arrêt tombe également dans cette période. Il a montré qu'aucun programme ne peut décider si un programme arbitraire donné (et son entrée respective) s'exécutera dans une boucle infinie. Avec

2Les logiques d'ordre supérieur sont des extensions de la logique des prédicats, dans laquelle non seulement des variables, mais aussi des symboles de fonction ou des prédicats peuvent apparaître comme termes dans une quantification. En effet, Gödel a seulement montré que tout système basé sur la logique des prédicats et capable de formuler l'arithmétique de Peano est incomplet.

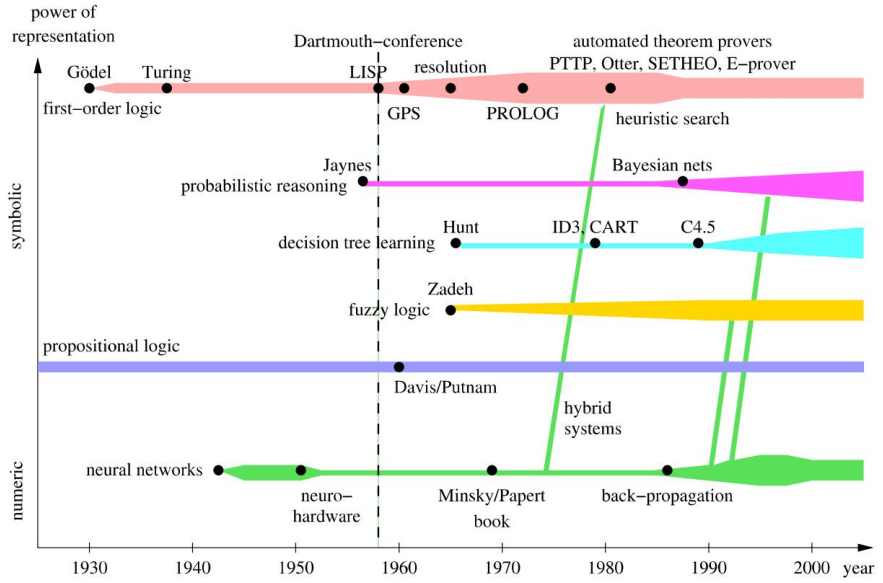


Fig. 1.3 Historique des différents domaines de l'IA. La largeur des barres indique la prévalence de l'utilisation de la méthode

ce Turing a également identifié une limite pour les programmes intelligents. Il s'ensuit, par exemple, qu'il n'y aura jamais de système universel de vérification de programme3.

Dans les années 1940, sur la base des résultats des neurosciences, McCulloch, Pitts et Hebb ont conçu les premiers modèles mathématiques de réseaux de neurones. Cependant, les ordinateurs de l'époque manquaient de puissance pour simuler des cerveaux simples.

1.2.2 La logique résout (presque) tous les problèmes

L'IA en tant que science pratique de la mécanisation de la pensée n'a bien sûr pu commencer qu'avec les ordinateurs programmables. Ce fut le cas dans les années 1950. Newell et Simon ont présenté Logic Theorist, le premier démonstrateur automatique de théorèmes, et ont ainsi montré qu'avec les ordinateurs, qui ne fonctionnent en fait qu'avec des nombres, on peut aussi traiter des symboles. A la même époque McCarthy introduit, avec le langage LISP, un langage de programmation spécialement créé pour le traitement des structures symboliques.
Ces deux systèmes ont été introduits en 1956 lors de la conférence historique de Dartmouth, considérée comme l'anniversaire de l'IA.

Aux États-Unis, LISP est devenu l'outil le plus important pour la mise en œuvre de systèmes d'IA à traitement de symboles. Par la suite, la règle d'inférence logique connue sous le nom de résolution s'est développée en un calcul complet pour la logique des prédicats.

3Cette déclaration s'applique à la « correction totale », ce qui implique une preuve d'exécution correcte ainsi qu'une preuve de terminaison pour chaque entrée valide.

Dans les années 1970, le langage de programmation logique PROLOG a été introduit en tant que Homologue européen de LISP. PROLOG offre l'avantage de permettre une programmation à l'aide de clauses Horn, un sous-ensemble de la logique des prédicats. Comme LISP, PROLOG a des types de données pour un traitement pratique des listes.

Jusque tard dans les années 1980, un esprit révolutionnaire dominait l'IA, en particulier parmi les de nombreux logiciens. La raison en était la série de réalisations impressionnantes dans traitement des symboles. Avec le projet Fifth Generation Computer Systems au Japon et le programme ESPRIT en Europe, d'importants investissements ont été consacrés à la construction de ordinateurs intelligents.

Pour les petits problèmes, démonstrateurs automatiques et autres systèmes de traitement de symboles fonctionnait parfois très bien. L'explosion combinatoire de l'espace de recherche a cependant défini une fenêtre très étroite pour ces succès. Cette phase de l'IA a été décrite dans [RN10] comme le « Regarde, maman, pas de mains ! » ère.

Parce que le succès économique des systèmes d'IA n'a pas répondu aux attentes, le financement pour la recherche sur l'IA basée sur la logique aux États-Unis a chuté de façon spectaculaire au cours des années 1980.

1.2.3 Le nouveau connexionnisme

Durant cette phase de désillusion, informaticiens, physiciens et Cognitifs les scientifiques ont pu montrer, à l'aide d'ordinateurs désormais suffisamment puissants, que les réseaux de neurones modélisés mathématiquement sont capables d'apprendre à l'aide exemples de formation, pour effectuer des tâches qui nécessitaient auparavant une programmation coûteuse. En raison de la tolérance aux pannes de ces systèmes et de leur capacité à reconnaître les modèles, des succès considérables sont devenus possibles, en particulier dans la reconnaissance des modèles. La reconnaissance faciale sur les photos et la reconnaissance de l'écriture manuscrite sont deux exemples d'applications. Le système Nettalk a pu apprendre la parole à partir de textes d'exemple [SR86]. Sous le nom de connexionnisme, une nouvelle sous-discipline de l'IA est née.

Le connexionnisme a explosé et les subventions ont afflué. Mais bientôt même ici la faisabilité les limites sont devenues évidentes. Les réseaux de neurones pourraient acquérir des capacités impressionnantes, mais il n'était généralement pas possible de saisir le concept appris dans des formules simples ou règles logiques. Les tentatives de combiner les réseaux de neurones avec des règles logiques ou la connaissance d'experts humains se sont heurtés à de grandes difficultés. De plus, aucune solution satisfaisante à la structuration et à la modularisation des réseaux a été trouvé.

1.2.4 Raisonnement dans l'incertitude

L'IA en tant que science pratique et axée sur des objectifs a cherché un moyen de sortir de cette crise. Un souhaitait unir la capacité de la logique à représenter explicitement les connaissances avec la force des réseaux de neurones à gérer l'incertitude. Plusieurs alternatives ont été suggérées.

Le raisonnement probabiliste le plus prometteur fonctionne avec des probabilités conditionnelles pour les formules de calcul propositionnel. Depuis lors, de nombreux systèmes de diagnostic et experts ont été construits pour des problèmes de raisonnement quotidien utilisant des réseaux bayésiens. Le succès des réseaux bayésiens tient à leur compréhensibilité intuitive, la

sémantique propre de la probabilité conditionnelle, et de la théorie des probabilités vieille de plusieurs siècles et mathématiquement fondée.

Les faiblesses de la logique, qui ne peut fonctionner qu'avec deux valeurs de vérité, peuvent être résolu par la logique floue, qui introduit pragmatiquement une infinité de valeurs entre zéro et un. Même si aujourd'hui encore son fondement théorique n'est pas totalement solide,
il est utilisé avec succès, en particulier dans l'ingénierie de contrôle.

Un chemin très différent a conduit à la synthèse réussie de la logique et des réseaux de neurones sous le nom de systèmes hybrides. Par exemple, des réseaux de neurones ont été utilisés pour apprendre l'heuristique pour la réduction de l'énorme espace de recherche combinatoire dans la découverte de preuves [SE90].

Les méthodes d'apprentissage par arbre de décision à partir des données fonctionnent également avec des probabilités. Des systèmes tels que CART, ID3 et C4.5 peuvent rapidement et automatiquement créer des des arbres de décision qui peuvent représenter des concepts de logique propositionnelle et être ensuite utilisés comme systèmes experts. Aujourd'hui, ils sont un favori parmi les techniques d'apprentissage automatique (Section 8.4).

Depuis environ 1990, l'exploration de données s'est développée en tant que sous-discipline de l'IA dans le domaine d'analyse de données statistiques pour l'extraction de connaissances à partir de grandes bases de données. Données le minage n'apporte pas de nouvelles techniques à l'IA, il introduit plutôt l'exigence d'utiliser de grandes bases de données pour acquérir des connaissances explicites. Une application avec un grand marché potentiel consiste à diriger les campagnes publicitaires des grandes entreprises sur la base de l'analyse de plusieurs millions d'achats par leurs clients. En règle générale, les techniques d'apprentissage automatique telles que car l'apprentissage par arbre de décision entre en jeu ici.

1.2.5 Agents distribués, autonomes et apprenants

L'intelligence artificielle distribuée, DAI, est un domaine de recherche actif depuis environ 1985. L'un de ses objectifs est l'utilisation d'ordinateurs parallèles pour augmenter l'efficacité de solutionneurs de problèmes. Il s'est avéré, cependant, qu'en raison du nombre élevé de calculs complexité de la plupart des problèmes, l'utilisation de systèmes "intelligents" est plus bénéfique que la parallélisation elle-même.

Une approche conceptuelle très différente résulte du développement d'agents logiciels autonomes et de robots destinés à coopérer comme des équipes humaines.
Comme pour les véhicules Braitenberg susmentionnés, il existe de nombreux cas dans lesquels un agent individuel n'est pas capable de résoudre un problème, même avec des ressources illimitées. Seule la coopération de nombreux agents conduit au comportement intelligent ou à
la solution d'un problème. Une colonie de fourmis ou une colonie de termites est capable d'ériger bâtiments d'une très grande complexité architecturale, malgré le fait qu'aucune fourmi comprend comment le tout s'emboîte. Ceci est similaire à la situation de l'approvisionnement en pain d'une grande ville comme New York [RN10]. Il n'y a pas d'agence centrale de planification pour le pain, mais plutôt des centaines de boulangers qui connaissent leur métier respectif. quartiers de la ville et faire cuire la quantité appropriée de pain à ces endroits.

L'acquisition active de compétences par les robots est un domaine passionnant de la recherche actuelle. Là sont des robots aujourd'hui, par exemple, qui apprennent indépendamment à marcher ou à effectuer diverses motricité liée au football (Chap. 10). Apprentissage coopératif de plusieurs robots pour résoudre des problèmes ensemble en est encore à ses balbutiements.

1.2.6 L'IA grandit

Les systèmes ci-dessus proposés par l'IA aujourd'hui ne sont pas une recette universelle, mais un atelier avec un nombre gérable d'outils pour des tâches très différentes. La plupart de ces outils sont bien développés et sont disponibles sous forme de bibliothèques logicielles finies, souvent avec des interfaces utilisateur pratiques. La sélection du bon outil et son utilisation judicieuse dans chaque cas individuel sont laissées au développeur d'IA ou à l'ingénieur de connaissances. Comme tout autre artisanat, cela nécessite une solide éducation, que ce livre est destiné à promouvoir.

Plus que presque toute autre science, l'IA est interdisciplinaire, car elle s'appuie sur des découvertes intéressantes dans des domaines aussi divers que la logique, la recherche opérationnelle, les statistiques, l'ingénierie de contrôle, le traitement d'images, la linguistique, la philosophie, la psychologie et la neurobiologie. En plus de cela, il y a le domaine de l'application particulière. Réussir à développer un projet d'IA n'est donc pas toujours aussi simple, mais presque toujours extrêmement excitant.

1.3 Agents

Bien que le terme d'agents intelligents ne soit pas nouveau pour l'IA, ce n'est que ces dernières années qu'il a pris de l'importance grâce à [RN10], entre autres. Agent désigne assez généralement un système qui traite des informations et produit une sortie à partir d'une entrée. Ces agents peuvent être classés de différentes manières.

En informatique classique, les agents logiciels sont principalement employés (Fig. 1.4 à la page 11). Dans ce cas, l'agent consiste en un programme qui calcule un résultat à partir de l'entrée de l'utilisateur.

En robotique, en revanche, des agents matériels (également appelés robots) sont employés, qui disposent en outre de capteurs et d'actionneurs (Fig. 1.5 à la page 11). L'agent peut percevoir son environnement grâce aux capteurs. Avec les actionneurs, il effectue des actions et modifie son environnement.

En ce qui concerne l'intelligence de l'agent, on distingue les agents réflexes, qui ne réagissent qu'à l'input, et les agents à mémoire, qui peuvent également inclure le passé dans leurs décisions. Par exemple, un robot moteur qui, grâce à ses capteurs, connaît sa position exacte (et l'heure) n'a aucun moyen, en tant qu'agent réflexe, de déterminer sa vitesse. Si, au contraire, il enregistre la position, à des pas de temps courts et discrets, il peut ainsi facilement calculer sa vitesse moyenne dans l'intervalle de temps précédent.

Si un agent réflexe est contrôlé par un programme déterministe, il représente une fonction de l'ensemble de toutes les entrées vers l'ensemble de toutes les sorties. Un agent à mémoire, en revanche, n'est en général pas une fonction. Pourquoi? (Voir exercice 1.5 page 14.) Les agents réflexes sont suffisants dans les cas où le problème à résoudre implique un processus décisionnel de Markov. Il s'agit d'un processus dans lequel seul l'état actuel est nécessaire pour déterminer la prochaine action optimale (voir Chap. 10).

Un robot mobile qui doit se déplacer de la salle 112 à la salle 179 dans un bâtiment effectue des actions différentes de celles d'un robot qui doit se déplacer à la salle 105. En d'autres termes, les actions dépendent du but. De tels agents sont dits basés sur des buts.

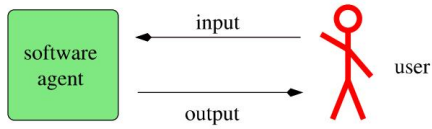
Tableau 1.1 Jalons du développement de l'IA de Gödel à aujourd'hui 1931 L'Autrichien

Kurt Gödel montre que dans la logique des prédicats du premier ordre, toutes les déclarations vraies sont dérivables [Göd31a]. Dans les logiques d'ordre supérieur, en revanche, il existe des énoncés vrais qui ne sont pas démontrables [Göd31b]. (Dans [Göd31b], Gödel a montré que la logique des prédicats étendue avec les axiomes de l'arithmétique est incomplète.)	
1937 Alan Turing pointe les limites des machines intelligentes avec le problème de l'arrêt [Tur37].	
1943 McCulloch et Pitts modélisent les réseaux de neurones et font le lien avec le propositionnel logique.	
1950 Alan Turing définit l'intelligence artificielle avec le test de Turing et écrit sur les machines d'apprentissage et les algorithmes génétiques [Tur50].	
1951 Marvin Minsky développe une machine à réseau neuronal. Avec 3000 tubes à vide, il simule 40 neurones.	
1955 Arthur Samuel (IBM) construit un programme d'apprentissage des échecs qui joue mieux que son développeur [Sam59].	
1956 McCarthy organise une conférence au Dartmouth College. Ici le nom Artificiel L'intelligence a été introduite pour la première fois.	
	Newell et Simon de l'Université Carnegie Mellon (CMU) présentent le Logic Theorist, le premier programme informatique de traitement de symboles [NSS83].
1958 McCarthy invente au MIT (Massachusetts Institute of Technology) le langage de haut niveau LISP. Il écrit des programmes capables de se modifier.	
1959 Gelernter (IBM) construit le démonstrateur de théorème de géométrie.	
1961 Le General Problem Solver (GPS) de Newell et Simon imite la pensée humaine [NS61].	
1963 McCarthy fonde le AI Lab à l'Université de Stanford.	
1965 Robinson invente le calcul de résolution pour la logique des prédicats [Rob65] (Section 3.5).	
1966 Le programme Eliza de Weizenbaum réalise un dialogue avec des personnes en langage naturel [Wei66] (Sect. 1.1.2).	
1969 Minsky et Papert montrent dans leur livre Perceptrons que le perceptron, un très simple réseau de neurones, ne peut représenter que des fonctions linéaires [MP69] (Sect. 1.1.2).	
1972 Le scientifique français Alain Colmerauer invente le langage de programmation logique PROLOG (Chapitre 5).	
	Le médecin britannique de Dombal développe un système expert pour le diagnostic des douleurs abdominales aiguës [dDLS+72]. Cela passe inaperçu dans la communauté de l'IA traditionnelle de l'époque (Section 7.3).
1976 Shortliffe et Buchanan développent MYCIN, un système expert de diagnostic des maladies infectieuses, capable de faire face à l'incertitude (Chap. 7).	
1981 Le Japon lance, à grands frais, le « Projet de cinquième génération » dans le but de construction d'une puissante machine PROLOG.	
1982 R1, le système expert de configuration des ordinateurs, fait économiser à Digital Equipment Corporation 40 millions de dollars par an [McD82].	
1986 Renaissance des réseaux de neurones à travers, entre autres, Rumelhart, Hinton et Sejnowski [RM86]. Le système Nettalk apprend à lire des textes à voix haute [SR86] (Chap. 9).	
1990 Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter introduisent la théorie des probabilités dans l'IA avec des réseaux bayésiens (Section 7.4). Les systèmes multi-agents deviennent populaires.	
1992 Le programme Tesauros TD-gammon démontre les avantages du renforcement apprentissage.	
1993 Initiative mondiale RoboCup pour construire des robots autonomes jouant au football [Roba].	

Tableau 1.1 (suite)

1995	A partir de la théorie de l'apprentissage statistique, Vapnik développe des machines à vecteurs supports, très importantes aujourd'hui.
1997	L'ordinateur d'échecs Deep Blue d'IBM bat le champion du monde d'échecs Gary Kasparov.
	Première compétition internationale RoboCup au Japon.
2003	Les robots de RoboCup démontrent de manière impressionnante ce dont l'IA et la robotique sont capables atteindre.
2006	La robotique de service devient un axe majeur de recherche en IA.
2010	Les robots autonomes commencent à apprendre leurs politiques.
2011	Le programme de compréhension du langage naturel et de réponse aux questions d'IBM « Watson » bat deux champions humains dans l'émission de quiz télévisée américaine « Jeopardy ! » (Section 1.4).

Fig. 1.4 Un agent logiciel avec interaction utilisateur



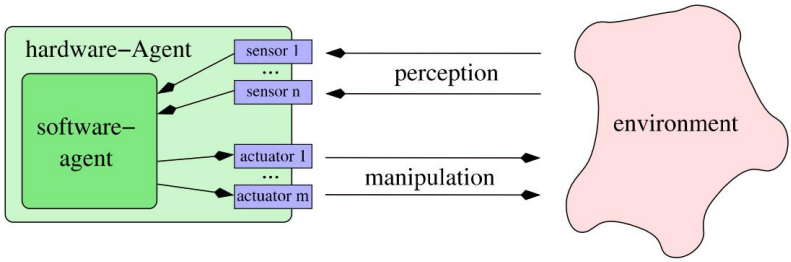


Fig. 1.5 Un agent matériel

Exemple 1.1 Un filtre anti-spam est un agent qui place les e-mails entrants dans des catégories recherchées ou indésirables (spam) et supprime tous les e-mails indésirables. Son objectif en tant qu'agent basé sur des objectifs est de mettre tous les e-mails dans la bonne catégorie. Au cours de cette tâche pas si simple, l'agent peut parfois faire des erreurs. Parce que son objectif est de classer correctement tous les e-mails, il tentera de faire le moins d'erreurs possible. Cependant, ce n'est pas toujours ce que l'utilisateur a en tête. Comparons les deux agents suivants. Sur 1 000 e-mails, l'agent 1 ne fait que 12 erreurs. L'agent 2, quant à lui, commet 38 erreurs avec les mêmes 1 000 e-mails. Est-ce donc pire que l'Agent 1 ? Les erreurs des deux agents sont présentées plus en détail dans le tableau suivant, appelé « matrice de confusion » :

Agent 1 :

		bonne classe	
		spam recherché 1	
le filtre anti-spam décide	recherché 189		
	spam 11 799		

Agent 2 :

		bonne classe	
		recherché spam	
le filtre anti-spam décide	recherché 200 38 spam		
	0 762		

L'agent 1 fait en fait moins d'erreurs que l'agent 2, mais ces quelques erreurs sont graves car l'utilisateur perd 11 e-mails potentiellement importants. Étant donné qu'il existe dans ce cas deux types d'erreurs de gravité différente, chaque erreur doit être pondérée avec le facteur de coût approprié (voir la section 7.3.5 et l'exercice 1.7 à la page 14).

La somme de toutes les erreurs pondérées donne le coût total causé par des décisions erronées. L'objectif d'un agent basé sur les coûts est de minimiser le coût des décisions erronées à long terme, c'est-à-dire en moyenne. Insecte. 7.3 nous allons nous familiariser avec le système de diagnostic LEXMED comme exemple d'agent basé sur les coûts.

De manière analogue, l'objectif d'un agent basé sur l'utilité est de maximiser l'utilité dérivée de décisions correctes à long terme, c'est-à-dire en moyenne. La somme de toutes les décisions pondérées par leurs facteurs d'utilité respectifs donne l'utilité totale.

Les agents d'apprentissage sont particulièrement intéressants pour l'IA , car ils sont capables de se modifier eux-mêmes à partir d'exemples de formation ou par des rétroactions positives ou négatives, de sorte que l'utilité moyenne de leurs actions augmente avec le temps (voir chap. 8).

Comme mentionné dans la Sect. 1.2.5, les agents distribués sont de plus en plus utilisés, dont l'intelligence n'est pas localisée dans un agent, mais ne peut être vue que par la coopération de nombreux agents.

La conception d'un agent est fortement orientée, avec son objectif, vers son environnement, ou alternativement son image de l'environnement, qui dépend fortement de ses capteurs. L'environnement est observable si l'agent connaît toujours l'état complet du monde. Sinon, l'environnement n'est que partiellement observable.
Si une action conduit toujours au même résultat, alors l'environnement est déterministe.
Sinon c'est non déterministe. Dans un environnement discret, seuls un nombre fini d'états et d'actions se produisent, alors qu'un environnement continu possède une infinité d'états ou d'actions.

1.4 Systèmes à base de connaissances

Un agent est un programme qui implémente une cartographie des perceptions aux actions. Pour les agents simples, cette façon de voir le problème est suffisante. Pour les applications complexes dans lesquelles l'agent doit pouvoir s'appuyer sur une grande quantité d'informations et est censé accomplir une tâche difficile, la programmation de l'agent peut être très coûteuse et peu claire sur la façon de procéder. Ici, l'IA fournit un chemin clair à suivre qui simplifiera grandement le travail.

Premièrement, nous séparons les connaissances du système ou du programme, qui utilise les connaissances pour, par exemple, tirer des conclusions, répondre à des questions ou élaborer un plan. Ce système est appelé mécanisme d'inférence. Les connaissances sont stockées dans une base de connaissances (KB). L'acquisition de connaissances dans la base de connaissances est appelée ingénierie des connaissances et repose sur diverses sources de connaissances telles que les experts humains, l'ingénieur des connaissances et les bases de données. Les systèmes d'apprentissage actif peuvent également acquérir des connaissances par l'exploration active du monde (voir chap. 10). Dans la Fig. 1.6 à la page 13 , l'architecture générale des systèmes à base de connaissances est présentée.

L'évolution vers une séparation de la connaissance et de l'inférence présente plusieurs avantages cruciaux. La séparation de la connaissance et de l'inférence peut permettre aux systèmes d'inférence d'être mis en œuvre d'une manière largement indépendante de l'application. Par exemple, c'est beaucoup