RAIRO. RECHERCHE OPÉRATIONNELLE

MALIK GHALLAB

Consistance, complétude et traduction optimale des tables de décision

RAIRO. Recherche opérationnelle, tome 12, n° 1 (1978), p. 61-84

<http://www.numdam.org/item?id=RO_1978__12_1_61_0>

© AFCET, 1978, tous droits réservés.

L'accès aux archives de la revue « RAIRO. Recherche opérationnelle » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/conditions). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 12, n° 1, février 1978, p. 61 à 84)

CONSISTANCE, COMPLÉTUDE ET TRADUCTION OPTIMALE DES TABLES DE DÉCISION (*)

par Malik GHALLAB (1)

Résumé. — *Une table de décision est une structure logique créant une correspondance entre un univers de décision et un ensemble de situations décrites par certaines conditions. Cet article aborde les problèmes de traduction d'une table en un code minimisant le « coût moyen de décision », avec vérification préalable de la consistance logique et de la complétude de la table.*

Parmi les méthodes de traduction connues, celle de Branch et Bound proposée par Reinwald et Soland, offre de nombreux avantages, mais son utilisation pratique est limitée à des tables de très petite dimension. L'algorithme exposé ici en est une extension par le modèle plus général des tables adopté, il pallie dans une large mesure aux inconvénients de la méthode initiale.

L'article débute par une modélisation des tables de décision et par une brève revue des méthodes de vérification et de traduction connues. Après la description de la procédure proposée, on termine sur quelques précisions concernant le programme de traduction écrit et ses performances.

1. DÉFINITIONS, PROPRIÉTÉS ET ALGORITHME DE VÉRIFICATION DES TABLES DE DÉCISION

1.1. Introduction

L'apparition de tables permettant de synthétiser à la fois les données et les conditions qui permettent de faire un choix parmi ces données, ne remonte qu'à une vingtaine d'années. Le développement d'outils informatiques de programmation à l'aide des tables de décision a suivi rapidement l'apparition de ces tables.

Les derniers processeurs écrits font apparaître un souci d'optimisation et un grand nombre d'algorithmes ont été proposés; ainsi en 1970, MacDaniel [20] recense plus de trente processeurs de tables différents, ce qui confirme la richesse de ce mode de représentation et son adaptation à la complexité croissante de la logique des problèmes actuellement envisagés.

Une table de décision est un ensemble de règles de décision, chaque règle créant une relation entre une suite d'actions et la valeur d'un ensemble de conditions.

(*) Reçu décembre 1976, révisé avril 1977.

(1) Laboratoire d'Automatique et d'Analyse des Systèmes, C.N.R.S., Toulouse.

Exemple :

C_1	1	1	0	0
C_2	1	0	1	0
a_1	*	*	*	*
a_2			*	*
a_3	*		*	*
a_4				*

Chaque colonne est une règle de décision, ainsi la colonne 3 de l'exemple s'interprète : si $\{ C_1$ est fausse et C_2 est vraie $\}$ alors exécuter les actions $\{ a_1, a_2, a_3 \}$.

La disposition tabulaire adoptée est celle de l'exemple. On peut dans certaines variantes ordonner la séquence d'actions correspondante à chaque règle.

On distingue fondamentalement deux types de tables :

– les tables à entrées limitées dans lesquelles les conditions sont des variables logiques à deux états;

– les tables à entrées étendues, où les conditions sont des variables discrètes à un nombre fini d'états.

L. I. Press [2] a établi un algorithme permettant de ramener toute table à une table à entrées limitées, on s'intéresse dans ce qui suit uniquement à ce dernier type.

1.2. Définitions et propriétés

1.2.1. *Conditions*

Soit $\mathscr{C} = \{ C_1, C_2, \ldots, C_j, \ldots, C_n \}$ un ensemble de n propositions logiques à valeur dans $\{ 0, 1 \}$: $C_j \to V(C_j) \in \{ 0, 1 \}$.

Les éléments de $\mathscr{C}$ sont des conditions qui décrivent un système dont l'état à un moment donné est représenté par un vecteur binaire S :

$$S = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_j, \ldots, \varepsilon_n) \quad \text{et} \quad \varepsilon_j \in \{ 0, 1 \} \quad \text{pour} \quad j = \overline{1, n}.$$

A l'état S du système les valeurs des conditions de $\mathscr{C}$ sont :

$$V(C_1) = \varepsilon_1; \ \ldots; \ V(C_n) = \varepsilon_n.$$

1.2.2. *Ensembles des α-cubes*

Soit Ω l'ensemble des 2^n sommets d'un n-cube, chaque élément de Ω est un vecteur binaire noté : $\omega_i = (\varepsilon_{i_1}, \ldots, \varepsilon_{i_j}, \ldots, \varepsilon_{i_n})$ avec $\varepsilon_{i_j} \in \{ 0, 1 \}$. Deux sommets de Ω sont adjacents s'ils ne diffèrent que par une seule composante.

On désigne par 1-cube l'union de 2 sommets adjacents : si $\omega_i = (\varepsilon_{i_1}, \ldots, \varepsilon_{i_n})$ et $\omega_k = (\varepsilon_{k_1}, \ldots, \varepsilon_{k_n})$ tel que $\varepsilon_{i_j} = \varepsilon_{k_j}$ pour $j = \overline{1, n}$; $j \neq l$ on note

$$\omega_i \vee \omega_k = (\varepsilon_{i_1}, \ldots, \varepsilon_{i_{l-1}}, \varphi, \varepsilon_{i_{l+1}}, \ldots, \varepsilon_{i_n}).$$

Exemple : $(1, 0, 1, 1, 0, 1) \vee (1, 0, 0, 1, 0, 1) = (1, 0, \varphi, 1, 0, 1)$.

De façon similaire, l'union de deux arêtes (ou 1-cube) adajacentes est un 2-cube, etc.

Soit $\mathscr{E}_0$ l'ensemble des 3^n α-cubes ($\alpha = 0, 1, \ldots, n$) définis sur un n-cube, chaque élément de $\mathscr{E}_0$ est un vecteur noté $W_k = (e_{k_1}, \ldots, e_{k_j}, \ldots, e_{k_n})$; $e_{k_j} \in \{ 0, 1, \varphi \}$. Un α-cube possède α composantes égales à φ, le remplacement de ces composantes de toutes les manières possibles par 1 ou 0, donne les 2^α sommets de l'α-cube. Tout α-cube est équivalent à l'union de ses 2^α sommets. Deux éléments de $\mathscr{E}_0$ sont dits indépendants si leur intersection est nulle. Inversement s'ils ont un ou plusieurs sommets communs, ils sont dépendants.

Le vecteur d'état du système décrit par $\mathscr{C}$ est nécessairement un des sommets du n-cube. Tout α-cube représente un ensemble de 2^α états potentiels du système qui sont ses sommets.

Soit $\mathscr{E} = \{ W_1, W_2, \ldots, W_r \}$ un sous-ensemble quelconque de $\mathscr{E}_0$. On associe à chaque élément W_k de $\mathscr{E}$, relativement à un état S du système une valeur logique $V(W_k)$ sur $\{ 0, 1 \}$:

— si S est un des sommets de l'α-cube $W_k \Rightarrow V(W_k) = 1$;

— sinon $V(W_k) = 0$.

1.2.3. *Règles de décision*

Soit $\{ a_1, a_2, \ldots, a_z \}$ un ensemble fini d'indices ou d'étiquettes chaque élément étant appelé une action; et soit $A_i = \{ a_{1_i}, \ldots, a_{l_i} \}$ une séquence ordonné non vide d'actions; $\mathscr{A} = \{ A_1, A_2, \ldots, A_p \}$ un ensemble de p séquences d'actions.

Soit $\mathscr{R}$ une surjection de $\mathscr{E}$ dans $\mathscr{A}$: $W_k \in \mathscr{E} \rightarrow \mathscr{R}(W_k) = A_i$. $\mathscr{R}$ induit une relation d'équivalence sur $\mathscr{E}$:

$$W_k \equiv W_l \quad \Leftrightarrow \quad \mathscr{R}(W_k) = \mathscr{R}(W_l),$$

à chaque séquence A_i de $\mathscr{A}$ correspond une classe d'équivalence π_i sur $\mathscr{E}$:

$$\pi_i = \{ W_k \in \mathscr{E}, \text{ tel que } \mathscr{R}(W_k) = A_i \}.$$

Tout couple (π_i, A_i) définit une règle de décision $R_i = (\pi_i, A_i)$: pour un état S du système on a :

— si $\exists\, k \in \overline{1, n}$ tel que $V(W_k) = 1$ et $W_k \in \pi_i \Rightarrow R_i$ consiste à choisir la séquence A_i;

— sinon R_i est non valide.

1.2.4. *Relation de dépendance sur* $\mathscr{C}$

Deux conditions C_j et C_l sont dépendantes, si dans le vecteur d'état S du système le couple $(\varepsilon_j, \varepsilon_l)$ ne peut prendre toutes les valeurs $\{(0,0);(0,1);(1,0);(1,1)\}$. Toute relation de dépendance logique sur $\mathscr{C}$ peut se mettre sous forme conjonctive, et donc se représenter par un α-cube W_i' de $\mathscr{E}$: la réalisation de tous les états représentés par les sommets de W_i' est logiquement exclue.

Exemple : $C_1 : x \geqq 5$, $C_2 : x \geqq 0$ on a $\{ C_1 \rightarrow C_2 \}$ qu'on écrit $\{ V(C_1).V(\bar{C}_2) = 0 \}$, cette relation s'exprime donc par le $(n-2)$ cube : $(1, 0, \varphi, \varphi, \ldots, \varphi)$. Soit $\mathscr{D} = \{ W_1', \ldots, W_d' \}$ le sous-ensemble de $\mathscr{E}_0$ traduisant *toutes* les relations de dépendances sur $\mathscr{C}$. Le couple $(\mathscr{C}, \mathscr{D})$ définit entièrement le système : son vecteur d'état S ne peut prendre que les valeurs de $(\Omega - \mathscr{D})$. On convient de : $\forall\, W_i' \in \mathscr{D}$; $V(W_i') = 0$.

1.2.5. *Table de décision*

Une table de décision T est définie formellement par le quintuplet : $T = \{ \mathscr{C}, \mathscr{D}, \mathscr{E}, \mathscr{A}, \mathscr{R} \}$;

– $\mathscr{C} = \{ C_1, \ldots, C_n \}$, ensemble des n conditions de la table;

– $\mathscr{D} = \{ W_1', \ldots, W_d' \}$, sous-ensemble de $\mathscr{E}_0$ exprimant toutes les relations de dépendance sur $\mathscr{C}$;

– $\mathscr{E} = \{ W_1, \ldots, W_r \}$, sous-ensemble de $\mathscr{E}_0$ traduisant tous les états pour lesquels une décision doit être prise;

– $\mathscr{A} = \{ A_1, \ldots, A_p \}$, $(p \leqq r)$ univers des décisions admissibles, chacune étant une séquence d'actions;

– $\mathscr{R}$ = surjection de $\mathscr{E}$ dans $\mathscr{A}$ qui induit sur $\mathscr{E}$ la partition $\Pi = \{ \pi_1, \ldots, \pi_p \}$.

Pour T donnée, aboutir à une décision dans un état S du système c'est trouver un α-cube W_k de $\mathscr{E}$ tel que $V(W_k) = 1$, la décision sera de choisir dans l'univers $\mathscr{A}$ la séquence $\mathscr{R}(W_k)$.

Exemple :

	W_1	W_2	W_3	W_4	W_5
C_1	1	1	φ	0	0
C_2	1	0	0	φ	1
C_3	φ	1	0	1	0
	A_1		A_2	A_3	

$\mathscr{C} = \{ C_1, C_2, C_3 \}$, $\mathscr{D} = \varnothing$,

$\mathscr{E} = \{ W_1, W_2, W_3, W_4, W_5 \}$,

$\mathscr{A} = \{ A_1, A_2, A_3 \}$,

$\mathscr{R}$ définit :

$$\pi_1 = \{W_1, W_2\}; \qquad \pi_2 = \{W_3\}; \qquad \pi_3 = \{W_4, W_5\}.$$

Si par exemple l'état du système est $S = (1, 1, 0)$ on a $V(W_1) = 1$ ce qui conduit à la décision : choix de A_1.

1.3. Simplification des tables de décision

1.3.1. *Réduction de $\mathscr{E}$*

Le but est d'aboutir à un nombre minimal de α-cubes dans chaque classe π_i de la partition Π de $\mathscr{E}$. La méthode est illustrée par l'exemple suivant : soit

$$W_1 = (1, 1, 0, 1, 1); \qquad W_2 = (1, 1, \varphi, 0, 1); \qquad W_3 = (1, 1, 1, 0, 0);$$
$$W_4 = (1, 0, 0, 1, 1); \qquad W_5 = (1, 0, 0, 0, 1);$$

tel que ces 5 α-cubes soient équivalents (appartiennent à la même classe) : $W_4 \vee W_5 = (1, 0, 0, \varphi, 1)$, par ailleurs $W_2 = W_2' \vee W_2''$ avec $W_2' = (1, 1, 1, 0, 1)$ et $W_2'' = (1, 1, 0, 0, 1)$ d'où $W_1 \vee W_2'' = (1, 1, 0, \varphi, 1)$ et finalement

$$(W_1 \vee W_2'') \vee (W_4 \vee W_5) = (1, \varphi, 0, \varphi, 1); \qquad W_2' \vee W_3 = (1, 1, 1, 0, \varphi).$$

On a réduit cette classe à deux α-cubes.

L'algorithme de Quine-McKlausky s'applique à cette réduction [19]. On remarque cependant, qu'il n'est pas possible en général de réduire chaque classe π_i à un unique α-cube. Une modélisation des tables [12] qui suppose dans tous les cas cette réduction possible est donc extrêmement restrictive.

1.3.2. *Règle-Autre*

C'est une convention d'écriture qui permet de créer implicitement une $(p+1)$ -ième classe π_{p+1}; associée à une séquence d'action A_{p+1} : $\mathscr{E}$ traduit une partie des états pour lesquels une décision doit être prise, tout état qui ne correspond pas à un α-cube de $\mathscr{E}$ correspond implicitement à un sommet de π_{p+1} d'où la relation : $\pi_{p+1} = \{(\Omega - \mathscr{D}) - \mathscr{E}\}$.

Cette relation permet d'expliciter la classe π_{p+1} donc de compléter $\mathscr{E}$ en $\mathscr{E} \cup \pi_{p+1}$ pour retrouver la définition de 1.2.5. Dans tout ce qui suit on ne fera aucune distinction pour les tables utilisant la convention « Règle-Autre ».

1.3.3. *Redondance dans $\mathscr{E}$*

Il y a redondance si deux α-cubes W_i et W_k appartiennent à une même classe et sont dépendants : leurs sommets communs sont redondants. On remarque que l'algorithme de réduction de Quine-McKlausky peut introduire une telle redondance.

1.3.4. *Redondance dans $\mathscr{C}$*

Une condition C_j de $\mathscr{C}$ est redondante, s'il existe une réduction de $\mathscr{E}$ dans laquelle tous les α-cubes aient pour j-ième composante φ. En effet pour T donnée, aboutir à une décision dans un état S, c'est trouver un α-cube W_k

tel que $V(W_k) = 1$. Si $e_{k_j} = \varphi_i$; ε_j l'état de la condition C_j n'intervient pas dans $V(W_k)$.

1.4. Consistance et complétude

1.4.1. *Définitions*

Une table est complète si pour tout état S du système, une règle de décision est valide. La table est consistante si cette règle est unique :

- si pour tout S donné, $\exists\, k \in \overline{1, n}$ tel que $V(W_k) = 1$; la table est complète;
- si pour tout S donné, $\exists\, k_1, k_2, \ldots, k_\alpha \in \overline{1, n}$ tel que

$$V(W_{k_1}) = V(W_{k_2}) = \ldots = V(W_{k_\alpha}) = 1$$

$$\Rightarrow \quad \mathscr{R}(W_{k_1}) = \mathscr{R}(W_{k_2}) = \ldots \mathscr{R}(W_k) = A_i \text{ unique};$$

la table est consistante.

Ces définitions entraînent :

T consistante $\Leftrightarrow \forall\, i$ et $k \in \overline{1, p}$: $\pi_i \cap \pi_k = \varnothing$ et $\pi_i \cap \mathscr{D} = \varnothing$;

T complète $\Leftrightarrow \mathscr{D} \cup \left(\bigcup_{i=1}^{p} \pi_i \right) = \Omega$.

1.4.2. *Algorithme de vérification des tables*

Initialisation : $\Omega_1 = \Omega - \mathscr{D}$.

Itération 1 : on vérifie $\pi_1 \subset \Omega_1$ et on définit $\Omega_2 = \Omega_1 - \pi_1$.

Itération m : on vérifie $\pi_m \subset \Omega_m$ si oui les m premières règles de décision $\{ (\pi_1, A_1), \ldots, (\pi_m, A_m) \}$ sont consistantes; on définit $\Omega_{m+1} = \Omega_m - \pi_m$.

Dernière itération : si $\pi_p \subset \Omega_p \Rightarrow T$ est consistante, si $\pi_p = \Omega_p$, T est complète, sinon $\Omega_{p+1} = \Omega_p - \pi_p$ constitue l'ensemble des états pour lesquels rien n'est spécifié et qu'il reste à répartir entre $\mathscr{E}$ et $\mathscr{D}$. Si l'option Règle-Autre est adoptée, T est nécessairement complète car $\Omega_{p+1} = \pi_{p+1}$ correspond à A_{p+1} sans ambiguité.

Le modèle de table et le formalisme adopté conduisent donc à un algorithme extrêmement simple, rapide à programmer et très efficace en comparaison notamment avec l'algorithme de King [8] qui ne permet pas de tester la complétude ni de distinguer un état ambigu d'une relation de dépendance sur $\mathscr{C}$.

2. TRADUCTION OPTIMALE DES TABLES DE DÉCISION

2.1. Généralités sur la traduction des tables

Traduire une table de décision T donnée c'est aboutir à un codage de T, appelé programme objet, qui une fois implanté sur calculateur, permettra à chaque interrogation de tester en partie ou en totalité les conditions de $\mathscr{C}$ et

d'aboutir à une décision de choix dans $\mathscr{A}$. L'avantage d'une traduction automatique d'une table de décision est de permettre :

– de tester au préalable la consistance et la complétude de la table;

– de refaire à moindre coût la traduction de la table après toute modification;

– et surtout d'envisager une optimisation du programme objet.

Il existe fondamentalement trois méthodes de traduction automatique.

2.1.1. *Méthode de Veinott* [4]

On teste successivement toutes les conditions de $\mathscr{C}$ et on incrémente convenablement un compteur suivant le résultat de chaque test. En fin de procédure, ce compteur contient l'adresse de la séquence à choisir dans $\mathscr{A}$. Cette méthode extrêmement simple est avantageuse pour une traduction manuelle, de plus elle se généralise au cas des tables à entrées étendues. Néanmoins elle nécessite de connaître à chaque décision tout le vecteur d'état S, ce qui est un inconvénient majeur.

2.1.2. *Méthode des masques* [5, 8, 14, 16]

On explore $\mathscr{E}$, α-cube par α-cube, et pour chaque élément W_k on ne teste que les conditions dont on ne connaît pas encore l'état et qui sont nécessaires pour le calcul de $V(W_k)$. Plusieurs règles ont été proposées pour ordonner les éléments de $\mathscr{E}$ en vue d'une « minimisation », mais la méthode reste peu performante et impose de tester en général beaucoup plus de conditions qu'il n'est besoin.

La troisième méthode, dite méthode de l'arbre binaire, est celle qui a été adoptée et développée sans la suite.

2.2. Arbre binaire et table de décision

2.2.1. *Arbre de décision traduisant une table*

Dans un arbre binaire, de chaque nœud partent deux branches, chaque branche pouvant soit être rattachée à un autre nœud, soit être une branche terminale, soit être une branche pendante. Un arbre binaire ne possédant pas de branche pendante est dit complet, sinon c'est un arbre partiel.

Un arbre de décision est un arbre binaire complet où chaque nœud représente une condition C_j de $\mathscr{C}$ et chaque branche terminale une séquence A_i de $\mathscr{A}$, en convenant qu'une branche gauche issue du nœud C_j est associée à la valeur $V(C_j) = 1$ et une branche droite à $V(C_j) = 0$, et que tout chemin reliant une branche terminale au sommet de l'arbre ne peut rencontrer plus d'une fois toute condition C_j de $\mathscr{C}$.

Soit $\mathscr{B} = \{ b_1, \ldots, b_s \}$ l'ensemble des branches terminales d'un arbre de décision, chaque élément de $\mathscr{B}$, $b_i = (e_{i_1}, \ldots, e_{i_j}, \ldots, e_{i_n})$ est un α-cube défini par :

— $e_{i_j} = 1$ si le chemin reliant la branche b_i au sommet de l'arbre passe par la branche gauche du nœud C_j,

— $e_{i_j} = 0$ si ce chemin passe par la branche droite de C_j.

— $e_{i_j} = \varphi$ si ce chemin ne rencontre pas C_j.

Si on note $\mathscr{R}' : \mathscr{B} \to \mathscr{A}$ la surjection qui à chaque branche terminale associe une séquence A_i de $\mathscr{A}$, on définit formellement un arbre de décision par : $t = \{ \mathscr{C}, \mathscr{B}, \mathscr{A}, \mathscr{R}' \}$

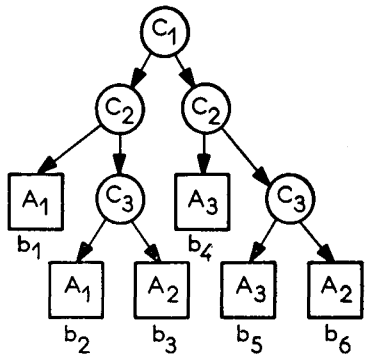
Un arbre de décision t traduit une table $T = \{ \mathscr{C}, \mathscr{D}, \mathscr{E}, \mathscr{A}, \mathscr{R} \}$ si pour tout couple $b_i \in \mathscr{B}$, $W_k \in \mathscr{E}$ on a

$$\{ b_i \text{ et } W_k \text{ dépendants} \Rightarrow \mathscr{R}'(b_i) = \mathscr{R}(W_k) \}.$$

Cette définition se justifie car si b_i et W_k sont dépendants, un de leurs sommets communs correspond à un état S pour lequel $V(W_k) = V(b_i) = 1$ donc W_k et b_i conduisent à la même décision de choix dans $\mathscr{A}$.

Exemple :

	W_1	W_2	W_3	W_4	W_5
C_1	1	1	φ	0	0
C_2	1	0	0	φ	1
C_3	φ	1	0	1	0
	A_1		A_2	A_3	



L'ensemble des branches terminales de cet arbre est :

$$\mathscr{B} = \{ b_1 = (1, 1, \varphi); b_2 = (1, 0, 1); b_3 = (1, 0, 0); b_4 = (0, 1, \varphi); b_5 = (0, 0, 1); b_6 = (0, 0, 0) \}$$

on vérifie par exemple que b_3 et b_6 dépendent de W_3, ils mènent tous à la même décision.

2.2.2. *Arbre partiel, branche pendante et sous-table*

Une branche pendante dans un arbre partiel sera caractérisée également par un α-cube $b_i = (e_{i_1}, \ldots, e_{i_j}, \ldots, e_{i_n})$:

— $e_{i_j} = 1$ ou 0, si la condition C_j se trouve sur le chemin reliant cette branche pendante au sommet de l'arbre;

$e_{i_j} = \varphi$, si C_j ne se trouve pas sur ce chemin.

Relativement à une table donnée, on associe à chaque branche pendante b_i une sous-table $T_i = \{ \mathscr{C}_i, \mathscr{D}_i, \mathscr{E}_i, \mathscr{A}_i, \mathscr{R}_i \}$ définie par :

$\mathscr{C}_i = \{ C_j \in \mathscr{C}$ tel que $e_{i_j} = \varphi \}$, les conditions de $\mathscr{C}$ qui ne figurent pas sur le chemin;

$\mathscr{E}_i = \{ W_k \in \mathscr{E}$ tel que b_i et W_k soient dépendants $\}$, de même pour $\mathscr{D}_i$;

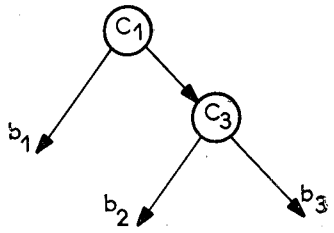
$\mathscr{R}_i$: restriction de $\mathscr{R}$ à $\mathscr{E}_i$;

$\mathscr{A}_i$: image de $\mathscr{E}_i$ par $\mathscr{R}_i$.

Exemple :

Relativement à la table de l'exemple précédent, la sous-table associée à la branche pendante $b_1 = (1, \varphi, \varphi)$ portera sur C_2 et C_3 et contiendra les α-cubes : $\{ W_1, W_2, W_3 \}$:

	W_1	W_2	W_3
C_2	1	0	0
C_3	φ	1	0
	A_1		A_2



On remarque qu'une sous-table associée à une branche pendante peut contenir une ou plusieurs conditions redondantes qu'il faut éliminer.

Si pour une branche pendante donnée b_i la sous-table correspondante t_i ne contient qu'une seule séquence A_k de $\mathscr{A}$, alors b_i est une branche terminale et $\mathscr{R}'(b_i) = A_k$.

2.2.3. *Traduction d'une table de décision en arbre de décision*

L'algorithme dû à Pollack [3] qui recherche un arbre de décision traduisant une table donnée, consiste à associer à une branche pendante une des conditions de la sous-table qui lui correspond et à créer un nouveau nœud. On complète ainsi un arbre partiel jusqu'à l'obtention d'un arbre complet :

1° Initialisation : choisir dans la table de décision une condition qui servira de sommet de l'arbre, prendre une des deux branches pendantes de ce sommet.

2° Construire la sous-table correspondant à cette branche pendante :

– si cette sous-table ne contient qu'une seule séquence d'actions A_i, transformer la branche pendante en branche terminale associée à A_i;

– sinon choisir dans la sous-table une condition, construire un nœud contenant cette condition et le rattacher à la branche pendante.

3° Si l'arbre partiel obtenu contient au moins une branche pendante en choisir une et retourner à l'étape 2° :

– sinon c'est un arbre complet traduisant la table.

Dans un arbre on peut aboutir à une décision en ne connaissant que partiellement le vecteur d'état du système : ceci constitue le principal avantage de cette méthode de traduction des tables.

L'arbre résultant de l'algorithme précédent n'est évidemment pas unique : à chaque sous-table on a le choix entre plusieurs conditions pour prolonger une branche pendante. Soit Θ l'ensemble des arbres traduisant une table de n conditions, et $|\Theta(n)|$ le cardinal de Θ. Un arbre traduisant une table de $(n+1)$ conditions est constitué d'une racine et de deux sous-arbres chacun traduisant une table de n conditions (si l'on exclut les redondances) :

$$|\Theta(n+1)| = (n+1).|\Theta(n)|^2 \qquad \text{or} \qquad |\Theta(1)| = 1$$

d'où

$$|\Theta(n)| = \prod_{k=0}^{n-1} (n-k)^{2^k}.$$

Bien entendu dans chaque sous-table il y a des conditions redondantes qu'on élimine, et les $|\Theta(n)|$ éléments de Θ ne sont tous distincts que si la table T contient $p = r = 2^n$ séquences d'actions distinctes.

2.3. Critères d'optimisation de la traduction d'une table

2.3.1. *Les trois critères d'optimisation*

Une traduction d'une table peut être estimée soit sur la base de « l'efficacité » de l'arbre traduisant la table (critère 1 et 2), soit sur celle du coût attaché à la recherche de cet arbre (critère 3).

Critère 1; coût moyen de décision : on suppose que le vecteur d'état du système n'est pas connu en permanence; à chaque fois qu'une décision doit être prise on teste en partie les conditions de $\mathscr{C}$, ce qui entraîne un certain coût, pondéré par la fréquence d'apparition de cet état dans le système.

Critère 2; coût de mémorisation de la table : la traduction d'une table abouti à un programme implanté sur calculateur de façon permanente; la taille mémoire nécessaire pour le stocker est un élément d'appréciation de la traduction.

Critère 3; coût de traduction : rechercher une solution optimale au sens du critère 1 ou 2 peut entraîner un coût de traduction élevé qui ne se justifie pas.

2.3.2. *Formulation du principal critère : coût moyen de décision*

Pour une table T, on définit une distribution de coût sur $\mathscr{C}$ et de probabilité sur $\mathscr{E}$:

– soit $\rho : \mathscr{C} \to R^+$; $\rho(C_j)$ est le coût d'évaluation de $V(C_j)$. On note pour simplifier $\rho_j = \rho(C_j)$;

– soit $\mu : \mathscr{E} \to [0, 1]$; $\mu(W_k)$ étant la probabilité d'apparition de l'un quelconque des états décrits par l'α-cube W_k. On suppose de plus que tous ces états sont équiprobables. La justification de cette hypothèse est due à la possibilité d'atteindre une décision sans déterminer à quel sommet d'un α-cube W_k correspond l'état actuel du système. Ces sommets ne sont donc pas observables expérimentalement et leur probabilité n'est pas accessible.

On convient que tous les états, logiquement exclus, décrits par $\mathscr{D}$ ont une probabilité d'apparition nulle : $\forall\, W_i' \in D,\ \mu(W_i') = 0$.

Soit t un arbre traduisant une table $T = \{ \mathscr{C}, \mathscr{D}, \mathscr{E}, \mathscr{A}, \mathscr{R} \}$ et soit $b_i \in \mathscr{B}$ une branche terminale quelconque de t, on note $\mu(b_i)$, la probabilité de passage par le chemin associé à b_i : si $\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_n}$, sont les sommets de l'α-cube b_i on a, compte tenu de ce qui précède :

$$\mu(b_i) = \mu(\omega_{i_1}) + \mu(\omega_{i_2}) + \ldots + \mu(\omega_{i_n}),$$

avec

$$\mu(\omega_{i_l}) = \frac{1}{2^{\alpha_k}} \mu(W_k)$$

si ω_{i_l} est un sommet de l'α_k-cube W_k. Pour l'arbre t, le coût moyen de décision $\psi(t)$, est le coût de passage depuis le sommet jusqu'à une branche terminale b_i, en testant toutes les conditions rencontrées sur le chemin et en pondérant ce coût par $\mu(b_i)$:

$$\psi(t) = \sum_{b_i \in \mathscr{B}} \left[\mu(b_i) . \sum_{j \in \mathscr{C}_i} \rho_j \right] \quad \text{avec} \quad \overline{\mathscr{C}}_i = \mathscr{C} - \mathscr{C}_i .$$

2.4. La procédure de séparation et d'évaluation progressive (P.S.E.P.)

2.4.1. *Intérêt de la P.S.E.P.*

Il existe plusieurs algorithmes pour l'optimisation de la traduction des tables de décision. Les algorithmes qui recherchent une solution optimale relativement aux critères 1 ou 2 [6, 7, 21, 23], sont basés sur des procédures énumératives, des procédures de séparation séquentielle ou progressive, ou de type programmation dynamique. Mais ils sont très coûteux relativement au critère 3 : la recherche d'une solution optimale est un problème NP-complet, (i. e. que le temps de calcul nécessaire pour faire cette recherche est une fonction exponentielle de la dimension de la table). Les algorithmes dits heuristiques, qui recherchent une solution sous-optimale [2, 3, 9, 10, 12, 15, 17, 18], plus performants relativement à ce critère 3, présentent néanmoins un inconvénient majeur : il n'est pas possible de situer, autrement que statistiquement, une solution sous-optimale par rapport à l'optimum, on ne peut même pas majorer l'écart entre les deux.

Une procédure P.S.E.P. permet d'accéder à une solution optimale en explorant moins d'éventualités que ne le ferait une procédure P.S.E.S. ou la programmation dynamique, elle permet de plus d'optimiser la traduction des tables relativement à un critère mixte qui serait une combinaison quelconque des critères 1 et 2. L'avantage le plus important de la P.S.E.P., qui lui est spécifique, est la possibilité d'atteindre une solution ε-optimale, sans connaître *a priori* ni une solution optimale, ni sa valeur dans le critère. Le temps de calcul, ainsi que la taille mémoire, nécessaire à la recherche d'une solution ε-optimale, décroissent quand ε croît, on pourra donc grâce à la P.S.E.P. tenir compte quantitativement du critère 3.

Reinwald et Soland ([6, 7]) ont proposé une procédure P.S.E.P. pour table entièrement développée. Bien qu'il soit possible de développer entièrement toute table, en remplaçant chaque α-cube de $\mathscr{E}$ par ses 2^{α} sommets, ce développement augmente considérablement la dimension de la table et restreint l'utilisation de l'algorithme de Reinwald et Soland : cela justifie les critiques qu'il a rencontré ([2]) et son peu de succès malgré les avantages théoriques importants de la P.S.E.P.

L'algorithme développé dans la suite, est une extension de celui de Reinwald et Soland, il permet l'utilisation de tous les avantages de la P.S.E.P. sur des tables quelconques, avec des performances comparables à celles des algorithmes heuristiques, même pour des tables de grandes dimensions.

2.4.2. *Ordre et partitions sur* Θ

Soit $\Theta = \{ t_1, t_2, \ldots, t_\gamma \}$ l'ensemble des arbres traduisant une table T donnée. On ordonne les nœuds d'un arbre quelconque t par l'ordre scriptural (la racine, les deux nœuds de rang 1 dans l'ordre transverse...). A chaque arbre correspond donc une liste ordonnée de nœuds de $\mathscr{C} = \{ C_1, \ldots, C_n \}$. L'ordre lexicographique permet de comparer les listes associées à deux arbres t et t', ce qui induit un ordre sur Θ.

Deux arbres t et t' sont dits équivalents modulo (m) s'ils ont leurs m premiers nœuds (au sens de l'ordre scriptural précédent) respectivement égaux. On note : $t \equiv t'$ (mod m). C'est une relation réflexive, transitive et symétrique.

Soit P_m la partition induite sur Θ par la relation ($\equiv$ mod m),

$$P_m = \{ \tau_{m_1}, \ldots, \tau_{m_\alpha} \}.$$

Chaque classe d'équivalence τ_{m_i} est un arbre partiel, c'est l'arbre constitué des m premiers nœuds communs aux éléments de cette classe.

Deux relations ($\equiv$ mod m) et ($\equiv$ mod m') induisent sur Θ deux partitions P_m et $P_{m'}$ telles que : si $m > m' \Leftrightarrow P_m$ est plus fine que $P_{m'}$.

([2]) On note que pour Shumacher et Sevcik [23], l'algorithme de Reinwald et Soland demanderait « plusieurs heures de calcul pour $n = 6$ (conditions) et des années et des siècle de calcul au-delà de $n = 6$ » !

En particulier P_{m+1} est la sous-partition directe de P_m. Si $\tau_{m_i} \in P_m$ les suivants directs de τ_{m_i} sont tous les éléments de P_{m+1} inclus dans τ_{m_i}. On note :

$S(m_i) = \{k$: tel que $\tau_k \in P_{m+1}$ et $\tau_k \subset \tau_{m_i}\}$; avec la relation $\bigcup_{k \in S(m_i)} \tau_k = \tau_{m_i}$.

Soit $\mathscr{T} = \bigcup_{m=0}^{M} P_m$, l'ensemble des arbres partiels de Θ; avec :

$P_0 = \{\tau_0\}$; τ_0 : arbre partiel vide ou classe contenant tous les éléments de Θ;

$P_M = \Theta$: la partition où chaque classe est un arbre complet.

L'ensemble $\mathscr{T}$ est ordonné par l'ordre lexicographique, pour deux éléments qui appartiennent à une même partition, et par l'ordre de finesse de leur partition respective dans le cas contraire.

2.4.3. *Fonction d'évaluation sur* $\mathscr{T}$

Dans ce paragraphe on définit, relativement à un critère $\psi : \Theta \to \mathbf{R}^+$ (2.3.2), une fonction d'évaluation $\mathscr{V} : \mathscr{T} \to \mathbf{R}^+$ tel que l'évaluation d'une classe τ_{m_i} de $\mathscr{T}$ soit un minorant du coût $\psi(t)$ pour tout t appartenant à τ_{m_i} :

$$\tau_{m_i} \in \mathscr{T} \to \mathscr{V}(\tau_{m_i}) \qquad \text{tel que} \quad \forall t \in \tau_{m_i} : \ \mathscr{V}(\tau_{m_i}) \leqq \psi(t) \tag{2.4.1}$$

2.4.3.1. *Calcul de la probabilité* q_j d'aboutir à une décision sans tester C_j.

Aboutir à une décision dans T, table consistante complète et sans redondance, c'est trouver l'α-cube $W_k \in \mathscr{E}$ tel que $V(W_k) = 1$, relativement à un état S donné. Pour connaître $V(W_k)$ il est inutile de tester toute condition C_j tel que $e_{k_j} = \varphi$. De même il est inutile de tester toute condition C_l dont la valeur $V(C_l)$ est déduite de la valeur d'autres conditions grâce à une relation de dépendance. Ainsi la probabilité d'aboutir à une décision dans T, sans tester *a priori* une condition C_j est la somme :

(i) des probabilités des α-cubes W_k ayant comme j-ième coordonnée $e_{k_j} = \varphi$;

(ii) des probabilités pondérées des α-cubes W_k qui interviennent dans une réduction où la j-ième composante serait φ (*cf.* § 1.3.1). Par exemple si W_k et W_i appartienne à la même règle :

$$W_k = (1, \varphi, 0, \varphi, 1)$$

$$= (1, 1, 0, 1, 1) \vee (1, 1, 0, 0, 1) \vee (1, 0, 0, 1, 1) \vee (1, 0, 0, 0, 1);$$

$$W_i = (1, 1, 1, 0, \varphi) = (1, 1, 1, 0, 1) \vee (1, 1, 1, 0, 0);$$

le 2e sommet de W_k et le 1er de W_i sont adjacents et se regroupent en $(1, 1, \varphi, 0, 1)$: dans la probabilité d'avoir à ne pas tester C_3 on aura les deux termes :

$$\frac{1}{4} \cdot \mu(W_k) + \frac{1}{2} \cdot \mu(W_i).$$

La pondération de $\mu(W_k)$ est le rapport $2^{\beta_{k_i}}/2^{\alpha_k}$:

- $2^{\beta_{k_i}}$: nombre de sommets de W_k ayant un adjacent dans W_i, donc β_{k_i} est le nombre de composantes de W_k telles que $e_{k_i} = e_{i_i} = \varphi$;
- 2^{α_k} : nombre total de sommets de l'α_k-cube W_k;

(iii) des probabilités pondérés des α-cubes W_k de $\mathscr{E}$ tel que il existe une relation de dépendance sur $\mathscr{C}$, $W_i' \in \mathscr{D}$; W_k et W_i' ayant deux ou plusieurs sommets adjacents suivants leur j-ième composante. Soit par exemple la relation de dépendance entre C_1, C_2, et C_3 : W_1' : (0, 1, 1, φ, ..., φ). Pour tout α-cube W_k tel $W_k = (0, 1, 0, e_{k_4}, \ldots, e_{k_n})$ il est inutile de tester C_3 car $V(C_1) = 0$ et $V(C_2) = 1$ implique par la relation $W_1' : V(C_3) = 0$. On remarque que W_k et W_1' ont au moins deux sommets adjacents suivant la 3e composante. La pondération de W_k est la même que précédemment.

On en déduit que la probabilité q_j d'aboutir à une décision dans T sans tester *a priori* la condition C_j est

$$q_j = \sum_k \delta_k^j . \mu(W_k) \tag{2.4.II}$$

- $\delta_k^i = 1$ pour un α-cube W_k tel que $e_{k_j} = \varphi$;
- $\delta_k^j = \sum_i 2^{\beta_{k_i}}/2^{\alpha_k}$ la somme étant étendu à l'ensemble des α-cube W_i équivalents à W_k et à l'ensemble des relations de dépendance W_i'; tel que W_k et W_i (ou W_i') aient au moins deux sommets adjacents suivant leur j-ième composante, $2^{\beta_{k_i}}$ est le nombre de sommets de W_k adjacents à des sommets de W_i (ou W_i') et 2^{α_k} nombre total de sommets de W_k;
- $\delta_k^j = 0$ si $e_{k_j} \neq \varphi$ et s'il n'existe aucun élément W_i (ou W_i') adjacent à W_k.

On remarque que si $q_j = 1$, la condition C_j est redondante dans T.

2.4.3.2. *Évaluation de* τ_0, *relation de récurrence sur* $\mathscr{V}$

Si l'on teste *a priori* une condition C_j de $\mathscr{C}$:

– l'espérance de dépense inutile est : $\rho_j . q_j$;

– l'espérance de dépense nécessaire est : $\rho_j . (1 - q_j)$; sommée sur toutes les conditions de la table, cette espérance est le coût moyen minimal pour aboutir à une décision, et constitue relativement au critère ψ un minorant du coût de tout arbre traduisant la table. On prendra ce minorant comme évaluation de τ_0, l'arbre partiel vide

$$\mathscr{V}(\tau_0) = \sum_{j=1}^{n} \rho_j (1 - q_j) \tag{2.4.III}$$

Soit τ_{1_j} l'arbre partiel réduit à son seul sommet C_j (i. e. la classe de tous les éléments de Θ ayant pour sommet C_j). Dans l'évaluation de τ_{1_j}, le coût ρ_j de C_j cessera d'être affecté d'une probabilité, car C_j est effectivement testée et

la dépense ρ_j effectuée pour tous les éléments de τ_{1_j}; les autres conditions C_l interviennent par l'espérance de dépense nécessaire : $\rho_l.(1-q_l)$; d'où :

$$\mathscr{V}(\tau_{i_j}) = \sum_{l \neq j} \rho_l.(1-q_l) + \rho_j$$

et utilisant (2.4.III)

$$\mathscr{V}(\tau_{1_j}) = \mathscr{V}(\tau_0) + \rho_j.q_j \qquad (2.4.\text{IV})$$

Soit τ_{m_α} un arbre partiel ayant m nœuds, et soit b_i une branche pendante dans τ_{m_α}. On défini (*cf.* 2.2.2), la sous-table $T_i = \{\mathscr{C}_i, \mathscr{D}_i, \mathscr{E}_i, \mathscr{A}_i, \mathscr{R}_i\}$ associée à la branche pendante b_i; la distribution de coût sur $\mathscr{C}_i$ est la restriction de celle définie sur $\mathscr{C}$; la distribution de probabilité sur $\mathscr{E}_i$ est

$$W_k \in \mathscr{E}_i \to \mu(W_k/b_i) = \frac{2^{\beta_{k_i}}}{2^{\alpha_k}}.\mu(W_k);$$

la pondération déduite de l'hypothèse d'équiprobabilité est le nombre de sommets de W_k figurants dans la sous-table T_i divisé par le nombre total de sommets de W_k. La probabilité de passage par la branche b_i est

$$\mu(b_i) = \sum_{W_k \in \mathscr{E}_i} \mu(W_k/b_i).$$

Si C_j est une des conditions de la sous-table T_i on définit formellement $q_j(b_i)$ de la même façon que q_j à été défini dans T par (2.4.II). La probabilité d'aboutir à une décision dans la sous-table T_i sans avoir à tester C_j est : $q_j(b_i)/\mu(b_i)$. Il est à noter que $q_j(b_i)$ tient compte implicitement de l'information contenue dans la branche b_i; c'est-à-dire les conditions précédemment testées ainsi que leur valeur logique.

Soit $\tau_{m+1,\,j}$ un suivant direct de τ_{m_α} obtenu en rajoutant la condition C_j comme $(m+1)$-ième nœud à la branche pendante b_i, si $\mathscr{V}(\tau_{m_\alpha})$ est l'évaluation de τ_{m_α}, on a la relation suivante ([3]) :

$$\mathscr{V}(\tau_{m+1,\,j}) = \mathscr{V}(\tau_{m_\alpha}) + \rho_j.q_j(b_i) \qquad (2.4.\text{V})$$

Cette relation de récurrence permet d'évaluer toute classe de $\mathscr{T}$ et de connaître un minorant du coût de ses éléments.

En plus de la condition (2.4.V) la fonction d'évaluation $\mathscr{V}$ vérifie les propriétés de :

– non décroissance (triviale à partir de 2.4.V);

– coïncidence : pour une classe τ réduite à un seul élément $\tau = \{ t \}$, on a $\mathscr{V}(\tau) = \psi(t)$; l'évaluation de la classe et le coût de l'arbre coïncident ([3]).

2.4.4. *Procédure* S.E.P., *recherche de la solution optimale*

Soit $\mathscr{V}^*$ une fonction d'élimination relativement au critère ψ qui représente le coût d'une solution connue *a priori*, non optimale :

$$\exists t \in \Theta \qquad \text{tel que} \quad \psi(t) < \mathscr{V}^*.$$

Pour toute classe τ_{m_α} tel que $\mathscr{V}(\tau_{m_\alpha}) > \mathscr{V}^*$, les éléments de τ_{m_α} auront tous un coût supérieur à $\mathscr{V}^*$ et la classe τ_{m_α} ne pouvant contenir de solution optimale, sera éliminée.

Soit $\mathscr{T}_0, \mathscr{T}_1, \ldots, \mathscr{T}_m, \ldots$, une suite de sous-ensemble de $\mathscr{T}$, la procédure S.E.P. lors de la m-ième itération est :

Étape 1 : On sélectionne dans $\mathscr{T}_m$ l'arbre partiel à évaluation minimale, soit τ_{m_0} cette classe :

– si $\tau_{m_0} = \{ \hat{t} \}$ se réduit à un arbre unique $\Rightarrow \hat{t}$ est l'optimum et $\forall\, t \in \Theta$:

$$\psi(\hat{t}) \leqq \psi(t).$$

La procédure est finie;

– si τ_{m_0} contient plusieurs éléments on poursuit la procédure.

Étape 2 : On sépare τ_{m_0} en construisant dessus la partition τ_{m+1}, soit $S(m_0)$ l'ensemble des indices des suivants directs de τ_{m_0}.

Étape 3 : On évalue les suivants directs de τ_{m_0} et on élimine ceux dont l'évaluation est supérieure à $\mathscr{V}^*$, soit $S'(m_0)$ définie par :

$$S'(m_0) = S(m_0) - \{ k; \text{ tel que } k \in S(m_0) \text{ et } \mathscr{V}(\tau_k) \geqq \mathscr{V}^* \}.$$

Étape 4 : On construit le $(m+1)$-ième sous-ensemble de $\mathscr{T}$:

$$\mathscr{T}_{m+1} = (\mathscr{T}_m - \{ \tau_{m_0} \}) \cup \Big(\bigcup_{k \in S'(m_0)} \tau_k \Big).$$

On reprend à l'étape 1.

La procédure s'initialise avec $\mathscr{T}_0 = \{ \tau_0 \}$ l'arbre partiel vide.

Cette procédure sélectionne bien un arbre optimal dans Θ car le principe de séparation vérifie les axiomes de :

– finitude : l'ensemble des classes τ_{m_0} est fini;

– complétude : $\bigcup_{k \in S(m_0)} \tau_k = \tau_{m_0}$;

([3]) Les démonstrations de la relation (2.4.V) et de la propriété de coïncidence sont disponibles chez l'auteur.

– d'arrêt : on s'arrête à une classe qui contient un seul élément. La fonction d'évaluation vérifie les propriétés de minoration : ($\mathscr{V}(\tau_{m_i}) < \psi(t)\ \forall\, t \in \tau_{m_i}$); et de coïncidence ([4]).

2.4.5. *Arrêt à une solution ε-optimale*

Soit $\varepsilon \in [0, 1]$ donné, on définit une fonction seuil $\delta_\varepsilon : \mathscr{T}^2 \to \{0, 1\}$, $\delta_\varepsilon(\tau_i, \tau_j) = 0$, si $\mathscr{V}(\tau_i) \leqq \mathscr{V}(\tau_j)$ et $[\mathscr{V}(\tau_j) - \mathscr{V}(\tau_i)]/\mathscr{V}(\tau_i) \leqq \varepsilon$.
Sinon, $\delta_\varepsilon(\tau_i, \tau_j) = 1$.

La fonction seuil signifie qu'une classe est préférée à une autre car elle présente une évaluation plus petite, mais que cette préférence ne dépasse pas un certain seuil. La procédure qui permet l'arrêt à une solution ε-minimale diffère de la précédente par une 5e étape :

Étape 5 : Soit $\tau_{(m+1)_0}$ l'élément présentant l'évaluation minimale sur $\mathscr{T}_{m+1}$, et soit $\tau_{(m+1)_1}$ celui qui présente l'évaluation minimale parmi les suivants directs de τ_{m_0} :

– si $\delta_\varepsilon(\tau_{(m+1)_0}, \tau_{(m+1)_1}) = 0 \Rightarrow$ on reprend la $(m+1)$-ième itération à l'étape 2, en remplaçant τ_{m_0} par $\tau_{(m+1)_1}$, c'est-à-dire son meilleur suivant direct. On continue ainsi à développer un même arbre partiel;

– si $\delta_\varepsilon(\tau_{(m+1)_0}, \tau_{(m+1)_1}) = 1 \Rightarrow$ on reprend à l'étape 1, donc en développant un autre arbre partiel.

Cette procédure s'arrête comme la précédente : si $\tau_{m_0} = \{\bar{t}\}$, alors $\bar{t}$ est une solution ε-minimale, et on est certain de :

$$\frac{\psi(\bar{t}) - \psi(\hat{t})}{\psi(\hat{t})} \leqq \varepsilon,$$

l'écart relatif entre $\bar{t}$ et l'optimum $\hat{t}$ est majoré par ε.

Pour $\varepsilon = 0$ cette procédure conduit à une solution optimale. Le programme de traduction des tables utilise cette procédure à deux reprises : une première fois pour déterminer $\mathscr{V}^*$ la fonction d'élimination en posant $\mathscr{V}^* = \psi(t^*)$ t^* étant une solution ε*-minimale, et une deuxième fois pour rechercher une solution ε-minimale, avec $\varepsilon^* > \varepsilon$. En effet toute classe dont l'évaluation excède $\psi(t^*)$ ne peut contenir de solution ε-minimale, son exploration est inutile.

REMARQUE : Le choix de l'ordre scriptural pour le développement d'un arbre partiel (étape 2 de l'algorithme) n'est pas fortuit. En effet la fonction d'évaluation choisie croît plus rapidement dans le cas d'un développement en largeur (ordre scriptural) que dans le cas d'un développement en profondeur. L'ordre scriptural améliore donc la convergence de l'algorithme.

([4]) La justiccation d'une telle procédure est faite dans le cas général par Roy [22], t. 2.

2.4.6. *Exemple d'application*

Reprenons l'exemple de 1.2.5 avec les distributions suivantes :

		W_1	W_2	W_3	W_4	W_5
ρ_j	$\mu(W_k)$	0,1	0,2	0,15	0,3	0,25
17	C_1	1	1	φ	0	0
9	C_2	1	0	0	φ	1
6	C_3	φ	1	0	1	0
		A_1		A_2	A_3	

on calcule q_1, q_2, q_3,

$$q_1 = \mu(W_3) = 0{,}15;$$

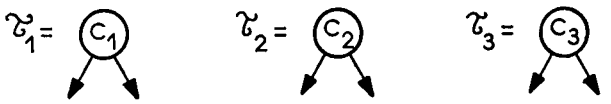
$$q_2 = \frac{1}{2}\mu(W_1)+\mu(W_2)+\mu(W_4) = 0{,}55;$$

$$q_3 = \mu(W_1)+\frac{1}{2}\mu(W_4)+\mu(W_5) = 0{,}5.$$

Initialisation : $\mathscr{T}_0 = \{\tau_0\}$, arbre partiel vide, on calcule $\mathscr{V}(\tau_0)$,

$$\mathscr{V}(\tau_0) = \sum_1^n \rho_j.(1-q_j) = 17.(1-q_1)+9.(1-q_2)+6.(1-q_3) = 21{,}5,$$

on développe τ_0, ses suivants directs sont



$\mathscr{V}(\tau_1) = \mathscr{V}(\tau_0)+\rho_1.q_1 = 21{,}5+2{,}55 = 24{,}05;$

$\mathscr{V}(\tau_2) = \mathscr{V}(\tau_0)+\rho_2.q_2 = 21{,}5+4{,}95 = 26{,}45;$

$\mathscr{V}(\tau_3) = \mathscr{V}(\tau_0)+\rho_3.q_3 = 21{,}5+3{,}0 = 24{,}5.$

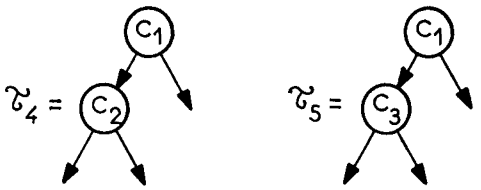
Itération 1 : $\mathscr{T}_1 = \{\tau_1, \tau_2, \tau_3\}$; τ_1 présente l'évaluation minimale on le développe, pour cela on construit la sous-table définie par $b_1 = (1, \varphi, \varphi)$:

	W_1	W_2	W_3
$\mu(W_k/b_1)$	0,1	0,2	0,075
C_2	1	0	0
C_3	φ	1	0
	A_1		A_2

$$q_2(b_1) = \frac{1}{2} \cdot \mu(W_1/b_1) + \mu(W_2/b_1) = 0{,}25;$$

$$q_3(b_1) = \mu(W_1/b_1) = 0{,}1;$$

d'où les suivants directs de τ_1 :



$\mathscr{V}(\tau_4) = 24{,}05 + 9 \times 0{,}25 = 26{,}30$; $\mathscr{V}(\tau_5) = 24{,}05 + 6 \times 0{,}1 = 24{,}65$.

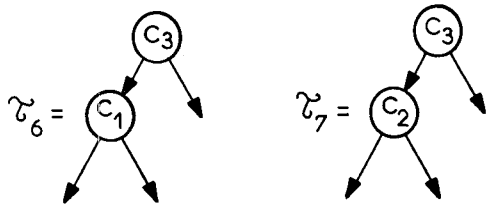
Itération 2 : $\mathscr{T}_2 = \{\tau_2, \tau_3, \tau_4, \tau_5\}$; on développe τ_3 qui présente l'évaluation minimale, la sous-table correspondant à la branche $b_2 = (\varphi, \varphi, 1)$ est

	W_1	W_2	W_4
$\mu(W_k/b_2)$	0,05	0,2	0,3
C_1	1	1	0
C_2	1	0	φ
	A_1		A_3

$$q_1(b_2) = 0,$$

$$q_2(b_2) = \mu(W_1/b_2) + \mu(W_2/b_2) + \mu(W_4/b_2) = 0{,}55;$$

d'où les suivants directs de τ_3 :



$\mathscr{V}(\tau_6) = 24{,}50$; $\mathscr{V}(\tau_7) = 24{,}5+9\times 0{,}55 = 29{,}45$.

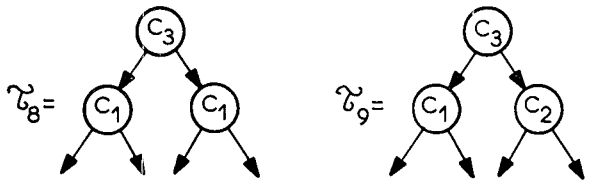
Itération 3 : $\mathscr{T}_3 = \{\tau_2, \tau_4, \tau_5, \tau_6, \tau_7\}$ c'est τ_6 qui présente la meilleure évaluation, la sous-table correspondant à la branche pendante $b_3 = (\varphi, \varphi, 0)$ est

	W_1	W_3	W_5
$\mu(W_k/b_3)$	0,05	0,15	0,25
C_1	1	φ	0
C_2	1	0	1
	A_1	A_2	A_3

$$q_1(b_3) = \mu(W_3/b_3) = 0{,}15;$$

$$q_2(b_3) = 0,$$

d'où les suivants directs de τ_6.

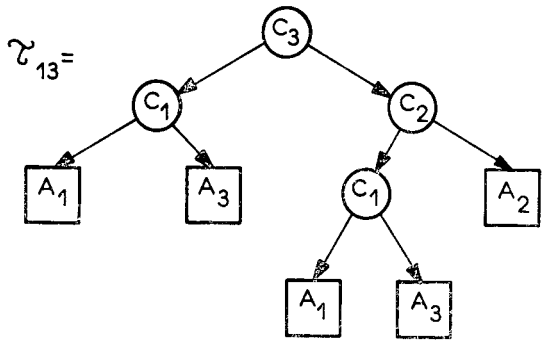


$\mathscr{V}(\tau_8) = 24{,}5+17\times 0{,}15 = 27{,}05$; $\mathscr{V}(\tau_9) = 24{,}5$.

Itération 4 : $\mathscr{T}_4 = \{\tau_2, \tau_4, \tau_5, \tau_7, \tau_8, \tau_9\}$; τ_9 présente la meilleure évaluation, son suivant direct est un arbre complet solution de l'optimisation. On

vérifie que pour $b_4 = (\varphi, 1, 0)$ on a : $q_1(b_4) = 0$; d'où :

$$\mathscr{V}(\tau_9) = \psi(\hat{t}) = 24{,}5$$



2.5. L'algorithme et sa programmation

2.5.1. *Le codage*

Si τ_{m_1} et τ_{m_2} sont tous deux suivants directs de $\tau_{m-1,k}$, toute l'information contenue dans $\tau_{m-1,k}$ se retrouve à la fois dans τ_{m_1} et dans τ_{m_2}. Le fait de remplacer $\tau_{m-1,k}$ par ses suivants directs introduit donc une grande redondance si l'on mémorise entièrement les suivants de chaque arbre partiel. Le codage retenu évite cette redondance et minimise l'information mémorisée.

2.5.2. *Améliorations apportées à la procédure*

En vue d'en accélérer la convergence, certaines améliorations ont été apportées à la procédure. La plus importante amélioration concerne la phase de sélection : on poursuit le développement d'un arbre partiel si son évaluation n'excède pas la fonction seuil δ_ε, sinon on choisit parmi tous les arbres partiels de $\mathscr{T}_m$, non pas celui qui présente l'évaluation minimale mais celui qui présente le meilleur compromis évaluation-niveau de développement, tout en n'excédant pas la fonction seuil δ_ε.

Dans le programme écrit, le critère retenu est $\psi(t)$ coût moyen de décision. Pour tenir compte du critère 2, taille mémoire occupée par l'arbre traduisant la table, une deuxieme fonction d'évaluation a été mise au point, la procédure calculant cette fonction, se rajoute au programme général sans aucune modification, et l'optimisation peut être faite suivant un critère mixte. En réalité cependant, ce critère de taille mémoire offre peu d'intérêt pour les raisons suivantes :

– il est possible de ne mémoriser chaque condition qu'une et une seule fois; aux nœuds de l'arbre on ne figurera que les adresses des conditions correspondantes, ainsi tous les arbres deviendront pratiquement équivalents au sens du 2e critère;

– on ne diminue réellement la taille mémoire occupée qu'en transformant l'arbre en graphe par supression des sous-arbres identiques. Pour ce faire, un algorithme « The duplicate sequence removal » a été proposé par Myers [33].

2.5 3. *Performances du programme*

Le programme de traduction des tables de décision mis au point, après lecture d'une table $T = \{ \mathscr{C}, \mathscr{D}, \mathscr{E}, \mathscr{A}, \mathscr{R}, \rho, \mu \}$ et de $\varepsilon \in [0, 1]$, vérifie la consistance et la complétude de T, en élimine toute redondance, et aborde la recherche de l'arbre ε-optimal traduisant T par la recherche d'une solution à 15 % de l'optimum dont le coût servira de fonction d'élimination. La solution ε-optimale est sortie récursivement sans forme graphique; il est possible également de l'obtenir sous forme d'instruction Fortran ou Cobol du type : IF (C_i) THEN k_{i_1}, k_{i_2}.

Ce programme a été testé sur des tables tirées de la littérature et jugées relativement complexes. Ainsi pour des tables contenant 10 conditions 20 actions et 40 α-cubes dans $\mathscr{E}$ et pour diverses distributions de coût et de probabilité, le programme fournit une solution optimale en 7,5 secondes en moyenne et une solution à 2 % de l'optimum en 4,4 secondes; la mémoire occupée ne dépassant pas 200 K, ceci sur un I.B.M. 370-168.

Ces performances contredisent totalement tout ce qui a été annoncé concernant la méthode de Branch et Bound : il a été possible de la mettre en œuvre pour des tables de grandes dimensions et le programme développé permet, relativement aux méthodes connues, un gain appréciable au niveau de la modélisation des tables, comme au niveau des performances quantitatives de la traduction.

Une récente publication [23] donne des informations concernant les performances de deux programmes différents :

– le premier, basé sur l'algorithme heuristique de Ganapathy fournit une solution sous-optimale – sans qu'on puisse la situer *a priori* par rapport à l'optimum – en un temps équivalent à celui que prend notre programme pour la recherche d'une solution à 2 % de l'optimum;

– le second, basé sur la programmation dynamique, permet d'accéder à une solution optimale avec des performances, analogues à celles de notre programme pour le temps de calcul, mais 4 fois plus élevé pour l'espace mémoire occupée.

Il est à remarquer de plus que ces deux programmes partent de tables entièrement développées, et ne permettent absolument pas de tenir compte de relations de dépendance entre les conditions.

En conclusion, le programme proposé par les possibilités de vérification de consistance, de complétude et d'élimination de redondance; offre un outil de mise au point des tables de décisions extrêmement complet. De plus il permet à l'utilisateur, au niveau de l'optimisation de la traduction des tables, de trouver un compromis entre le degré d'optimisation souhaité et la dépense

qu'entraîne cette optimisation, ceci est d'autant plus important que les tables traitées évoluent fréquemment aussi bien au niveau de leur structure logique, qu'au niveau des distributions de coût ou de probabilité.

REMERCIEMENTS

L'auteur tient tout particulièrement à remercier M. H. Gallaire, directeur du D.E.R.I.-C.E.R.T. pour ses nombreuses suggestions.

BIBLIOGRAPHIE

1. H. W. KIRK, *Use of Decision Table in Computer Programming*, Com. A.C.M., janvier 1965.
2. L. I. PRESS, *Conversion of Limited Entry Decision Table to Computer Programme*, Com. A.C.M., juin 1965.
3. S. U. POLLACK, *Conversion of Limited Entry Decision Table to Computer Programme*, Com. A.C.M., novembre 1965.
4. C. G. VEINOTT, *Programming Decision Table in Frotran, Cobol, Agol*, Com. A.C.M., janvier 1966.
5. P. J. H. KING, *Conversion of Decision Table by Rule Mask Techniques*, Com. A.C.M., novembre 1966.
6. REINWALD et SOLAND, *Conversion of Limited Entry Decision Table to an Optimal Computer Programme 1: Minimum Average Processing Time*, Com. A.C.M., juillet 1966.
7. REINWALD et SOLAND, *Conversion of Limited Entry Decision Table to an Optimal Computer Programme 2: Minimum Storage Requirement*, Com. A.C.M., octobre 1967.
8. P. J. H. KING, *Ambiguity in Limited Entry Decision Table*, Com. A.C.M., octobre 1968.
9. MUTHUKRISHNAN et RAJARAMAN, *On the Conversion of Decision Table to Computer Programme*, Com. A.C.M., juillet 1970.
10. K. SHWAYDER, *Conversion of Limited Entry Decision Table to Computer Programme: a Proposed Modification to Pollack's Algorithm*, Com. A.C.M., février 1971.
11. POLLACK, HICKS et HARRISON, *Decision Table, Theory and Pratice*, Wiley-Interscience, 1974.
12. YASUI, *Conversion of Decision Table into Decision Trees*, N.T.I.S., février 1972.
13. MYERS, *Compliling Optimized Code from Decision Table*, I.B.M. Journal, novembre 1972.
14. G. DATHE, *Conversion of Decision Tables by Rules Mask-Method without Rule Mask*, Com. A.C.M., octobre 1972.
15. M. VERHELST, *The Conversion of L.E.D.T. to Optimal or near Optimal Flowcharts*, Com. A.C.M., novembre 1972.

16. P. J. H. KING et R. G. JOHNSON, *Comments of the Use of Ambiguous Decision Table and their Conversion to Computer Programme*, Com. A.C.M., mai 1973.
17. S. GANAPATHY et V. RAJARAMAN, *The Information Theory applied to the Conversion of Decision Table to Computer Programme*, Com. A.C.M., septembre 1973.
18. K. SHWAYDER, *Extending the Information Theory Approach to Converting Limited Entry Decision Table to Computer Programme*, Com. A.C.M., septembre 1974.
19. K. SHWAYDER, *Combining Decision Rules in Decision Tables*, Com. A.C.M., août 1975.
20. H. MCDANIEL, *Decision Table Software*, Brandon systems press, 1971.
21. LEMAITRE, *Réalisations optimales et heuristiques de questionnaires*, Thèse, U.P.S., Toulouse, 1975.
22. B. ROY, *Algèbre moderne et théorie des graphes*, t. I et II. Dunod, 1970.
23 H. SCHUMACHER et K. C. SEVCIK, *The Synthetic Approach to the Decision Table Conversion*, Com. A.C.M., juin 1976.