CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision

(Université de Paris-Dauphine)

Unité Associée au C.N.R.S. n° 825

UN ALGORITHME LINEAIRE EN TEMPS ESPERE POUR DEDUIRE TOUTES LES CONCLUSIONS LOGIQUES INDUITES PAR UN ENSEMBLE D'INEGALITES BOOLEENNES

CAHIER N° 60

mai 1985

P. HANSEN

B. JAUMARD (*)

M. MINOUX (*)

(*) Centre National d'Etudes des Télécommunications, Issy-les-Moulineaux.

SOMMAIRE

	Pages
ABSTRACT	I
RESUME	II
1. INTRODUCTION	1
2. CONTRADICTION ET IDENTIFICATION	3
3. FIXATION	4
4. EXPERIENCES NUMERIQUES	9
REFERENCES	12
ANNEXES	I
ANNEXE I : Un exemple	II
ANNEXE II : Programme de l'algorithme CA	VII
ANNEXE III : Programme de l'algorithme LCA	X

A LINEAR EXPECTED-TIME ALGORITHM FOR DERIVING ALL LOGICAL CONCLUSIONS IMPLIED BY A SET OF BOOLEAN INEQUALITIES

ABSTRACT

Consider a set R of m binary relations on a set of n boolean variables. R may imply a contradiction, the fixation of some variables at 0 or at 1 and/or the identification of some pairs of variables in direct and complemented form. An O(n) expected-time algorithm is given for the derivation of all such logical conclusions. Computational experiments with problems involving up to 2000 variables are reported on. The proposed algorithm is more than 100 times faster than previous methods when $n \geq 100$.

Key-words : Boolean inequalities, implication graph, depth-first search, strongly connected components, transitive closure, 0-1 programming.

UN ALGORITHME LINEAIRE EN TEMPS ESPERE POUR DEDUIRE TOUTES LES CONCLUSIONS LOGIQUES INDUITES PAR UN ENSEMBLE D'INEGALITES BOOLEENNES

RESUME

Considérons un ensemble R de m relations binaires sur un ensemble de n variables booléennes. De R on peut déduire une contradiction, la fixation de variables à 0 ou à 1 et/ou l'identification de paires de variables sous forme directe ou complémentée. On propose un algorithme en O(n) en temps espéré pour la déduction de telles conclusions logiques. Des expériences de calcul sont présentées sur des problèmes comprenant jusqu'à 2000 variables. L'algorithme proposé est plus de 100 fois plus rapide que les méthodes précédentes lorsque $n \geq 100$.

Mots-clés : Inégalités booléennes, graphe d'implication, recherche en profondeur d'abord, composantes fortement connexes, fermeture transitive, programmation 0-1.

1 - INTRODUCTION

Soit R un ensemble de m relations binaires définies sur un ensemble X de n = |X| variables booléennes. Pour tout couple de variables $(x_i, x_k) \in X.X$, ces relations peuvent se mettre sous la forme d'égalités ou d'inégalités :

$$x_i \geq x_k \Leftrightarrow \bar{x}_k \geq \bar{x}_i \Leftrightarrow \bar{x}_i x_k = 0 \Leftrightarrow x_i \vee \bar{x}_k = 1 \qquad (1)$$

$$x_k \geq x_i \Leftrightarrow \bar{x}_i \geq \bar{x}_k \Leftrightarrow \bar{x}_k x_i = 0 \Leftrightarrow x_k \vee \bar{x}_i = 1 \qquad (2)$$

$$\bar{x}_i \geq x_k \Leftrightarrow \bar{x}_k \geq x_i \Leftrightarrow x_i x_k = 0 \Leftrightarrow \bar{x}_i \vee \bar{x}_k = 1 \qquad (3)$$

$$x_k \geq \bar{x}_i \Leftrightarrow x_i \geq \bar{x}_k \Leftrightarrow \bar{x}_k \bar{x}_i = 0 \Leftrightarrow x_k \vee x_i = 1 \qquad (4)$$

où v représente la somme booléenne.

A partir de R, on peut tirer des conclusions logiques d'un des types suivants :

a) Contradiction : une variable x_i doit prendre les valeurs 0 et 1 ;

b) Fixation : une variable x_i doit prendre la valeur 0 (ou doit prendre la valeur 1) ;

c) Identification : deux variables x_i, x_k doivent prendre la même valeur (ou doivent prendre des valeurs complémentaires).

Une manière d'obtenir toutes les conclusions logiques engendrées par R (cf Hammer et Nguyen [8], Hansen [9], Guignard et Spielberg [6], Johnson et Padberg [11]) consiste à construire d'abord la fermeture transitive $\hat{R}$ de R, soit l'ensemble des relations binaires déduites de R sur X par transitivité, puis à examiner toutes les paires de variables (x_i, x_k) en remarquant que :

a) Une contradiction intervient si et seulement si il existe un couple d'indices (i,k) (avec éventuellement i = k), tel que les relations (1) à (4) appartiennent à $\hat{R}$;

b) Si aucune contradiction n'intervient, la variable x_i doit prendre la valeur 0 (respectivement la valeur 1) si et seulement si il existe k, (avec éventuellement i = k), tel que (2) et (3) (respectivement (1) et (4)) appartiennent à la fois à $\hat{R}$;

c) Deux variables non fixées x_i et x_k doivent prendre la même valeur (respectivement des valeurs complémentaires) si et seulement si (1) et (2) (respectivement (3) et (4)) appartiennent à la fois à $\hat{R}$.

Déterminer $\hat{R}$ nécessite $0(n^3)$ opérations dans le pire cas avec une généralisation de l'algorithme de fermeture transitive de Roy-Warshall ([15], [18]) (voir Hansen [9], Johnson et Padberg [11]) ; cette complexité peut être réduite à $(n^{2.49})$ en exploitant l'équivalence entre le produit de matrices et le calcul de la fermeture transitive (voir Fisher et Meyer [5], Munro [13]) et les récentes améliorations (Coppersmith et Winograd [3]) de la méthode de multiplication rapide de matrices de Strassen [16].

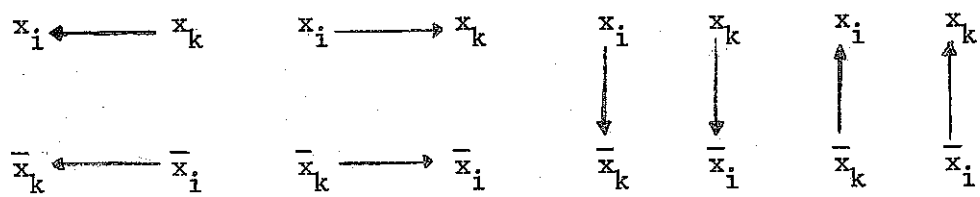
Cependant, les conclusions logiques peuvent être obtenues plus économiquement sans déterminer complètement $\hat{R}$. Dans le paragraphe suivant, nous montrerons que 0(m) opérations suffisent dans le pire cas pour détecter une contradiction et toutes les identifications de variables non fixées.

Dans le paragraphe 3, nous présentons un algorithme permettant d'obtenir toutes les conclusions logiques avec une complexité en temps espéré de 0(n). Le paragraphe 4 résume les expériences de calcul ; on constate que la complexité annoncée est observée même pour des valeurs modérées de m et de n.

Les conclusions logiques sont utiles dans les algorithmes de programmation linéaire et quadratique en variables 0-1 (cf Breu et Burdet [2] ; Crowder, Johnson et Padberg [4], Hammer et Hansen [7] et les références citées ci-dessus).

2- CONTRADICTION ET IDENTIFICATION

Comme dans Aspvall, Plass et Tarjan [1], associons à R un graphe d'implication $G = (X \cup \bar{X}, U)$ tel que : les 2n sommets correspondent aux variables $x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$; deux arcs orientés joignent les sommets correspondant aux littéraux des membres de droite dans les inégalités (1) - (4) aux littéraux des membres de gauche pour chacune des relations de R (voir figure 1).



Type (1)	Type (2)	Type (3)	Type (4)
$x_i \geq x_k$	$x_k \geq x_i$	$\bar{x}_k \geq x_i$	$x_i \geq \bar{x}_k$
$\bar{x}_k \geq \bar{x}_i$	$\bar{x}_i \geq \bar{x}_k$	$\bar{x}_i \geq x_k$	$x_k \geq \bar{x}_i$

Figure 1 : Représentation des relations dans le graphe d'implication

Chacun des arcs ainsi défini est tel que si le littéral correspondant au sommet extrémité initiale est égal à 1 alors il en est de même pour le littéral correspondant au sommet extrémité finale.

Comme l'ont montré Aspvall Plass et Tarjan, R est compatible si et seulement si aucune composante fortement connexe de G ne contient à la fois x_k et $\bar{x}_k$ pour $k \in \{1, 2, \ldots, n\}$. De plus, la condition pour l'identification d'une paire donnée de littéraux du paragraphe 1 est équivalente à l'appartenance des 2 sommets correspondant à ces littéraux à la même composante fortement connexe de G. Ainsi, déterminer les composantes fortement connexes permet de détecter une contradiction éventuelle et de faire les identifications de littéraux. La méthode bien connue de Tarjan fondée sur une recherche en profondeur d'abord peut être utilisée à cet effet, et nécessite O(m) calculs. Elle peut être accélérée d'un facteur constant dans le cas moyen, en détectant une contradiction dès que possible et en interrompant l'exécution lorsqu'elle est trouvée : c'est-à-dire chaque fois que l'on a déterminé une composante fortement connexe qui contient à la fois x_k et $\bar{x}_k$ pour k donné ; cette vérification peut être faite en respectant la complexité en O(m) dans le pire cas.

3- FIXATION

Il semble qu'on ne dispose à l'heure actuelle d'aucun algorithme pour détecter les fixations de variables d'un ensemble donné induites par R, nécessitant dans le pire cas moins de calculs que la détermination de la fermeture transitive de G. Cependant, si les m relations de R sont supposées choisies au hasard avec une égale probabilité parmi les relations binaires (1) - (4), un algorithme de complexité très faible en temps espéré peut être obtenu. Les règles de cet algorithme, que l'on appellera algorithme de conclusions logiques (LCA), sont les suivantes :

Etape 1. Construction du graphe d'implication et détermination de contradictions éventuelles

$R_o = \emptyset$;
Tant que $R_o \neq R$ faire
  Si $|R_o| \leq n$ alors poser $D = R/R_o$
    sinon D est égal à un ensemble de n relations distinctes choisies dans R_o ;
  Fsi
  Utiliser la recherche en profondeur d'abord pour trouver les composantes fortement connexes de $G = (X \cup \bar{X}, U_o)$ où U_o est l'ensemble des arcs correspondant à R_o. Si une composante fortement connexe de G contient pour un k donné, $k \in \{1, 2, \ldots, n\}$ à la fois x_k et $\bar{x}_k$, s'arrêter.
Fin-tant-que

Etape 2. Identification et graphe réduit

Pour toute paire de variables (x_i, x_k) (respectivement $(x_i, \bar{x}_k)$) appartenant à la même composante fortement connexe de G, identifier x_i et x_k (respectivement x_i et $\bar{x}_k$).

Construire le graphe réduit $G_R = (X_R, U_R)$ de G, où par définition

i) les sommets $x_{R_i} \in X_R$ sont en bijection avec les ensembles de sommets définis par les composantes fortement connexes de G,

ii) U_R contient un arc (x_{R_i}, x_{R_k}) si et seulement si G contient un arc joignant un sommet de la composante fortement connexe correspondant à x_{R_i} à un sommet de la composante fortement connexe correspondant à x_{R_k}.

Etape 3. Fermeture transitive du graphe réduit

Déterminer la fermeture transitive $\hat{G}_R$ de G_R (avec l'algorithme de Purdom [14] par exemple ou une version améliorée décrite dans Jaumard et Minoux [10]).

Etape 4. Fixation de variables

Pour chaque sommet X_{R_j} de $\hat{G}_R$, considérer la liste des variables associées à ce sommet et aux sommets qui sont les prédécesseurs de ce sommet dans $\hat{G}_R$. Si cette liste contient pour $k \in \{1, 2, \ldots, n\}$ à la fois x_k et $\bar{x}_k$, fixer les variables associées à x_{R_j} à 1 si elles apparaissent sous forme directe et à 0 si elles apparaissent sous forme complémentée.

Fin.

Proposition 1

L'algorithme LCA détecte toutes les contradictions et/ou les fixations de variables induites par R.

Démonstration

La détection d'une contradiction dans l'étape 1 résulte du résultat de Aspvall, Plass et Tarjan [1] discuté au paragraphe 2, quand $R = R_o$.

La variable x_i doit être fixée à 1 si et seulement si $\hat{R}$ contient les relations $x_i \geq \bar{x}_i$, ou à la fois les relations $x_i \geq x_k$ et $x_i \geq \bar{x}_k$ pour un k donné ; x_i doit être fixé à 0 si et seulement si $\hat{R}$ contient les relations $\bar{x}_i \geq x_i$, ou à la fois les relations $x_k \geq x_i$ et $\bar{x}_k \geq x_i$ pour un k donné. Comme le dernier cas est équivalent à $\bar{x}_i \geq \bar{x}_k$ et $\bar{x}_i \geq x_k$, on devra seulement rechercher si la même variable apparaît sous forme directe et complémentée dans une inégalité ou dans le membre de droite de 2 inégalités induites avec le même littéral dans le membre de gauche. Cela revient à tester si $\hat{G}$, la fermeture transitive de G, contient les arcs $(\bar{x}_i, x_i)$ ou $(x_i, \bar{x}_i)$, ou à la fois (x_k, x_i) et $(\bar{x}_k, x_i)$, ou à la fois $(x_k, \bar{x}_i)$ et $(\bar{x}_k, \bar{x}_i)$. On vérifie facilement que l'étape 4 réalise l'équivalent dans $\hat{G}_R$. ∎

Intéressons nous maintenant au comportement de l'algorithme en temps espéré.

Proposition 2

L'algorithme LCA nécessite un nombre espéré de calculs en $O(n)$.

Démonstration

Le point principal est de montrer qu'il existe une constante positive c (assez grande) telle que la probabilité pour que cn relations choisies au hasard n'impliquent pas de contradiction, décroît exponentiellement avec n. Ceci sera établi en plusieurs étapes :

a) Il existe des constantes positives k_1, ℓ_1 et c_1 tels que la probabilité qu'un ensemble de cn relations choisies au hasard ne contiennent pas $c_1 n$ relations du type (1) et (2), ainsi que $c_1 n$ relations du type (3) et (4) n'excèdent pas $k_1 e^{-\ell_1 n}$ pour n assez grand.

En effet, soit B_{cn} la variable aléatoire (de loi binomiale) égale au nombre de relations du type (1) et (2) parmi les cn relations données. Pour n grand, la distribution de la variable $N_{01} = \frac{B_{cn} - \frac{1}{2} cn}{\sqrt{cn\ 1/2\ (1-1/2)}}$ peut être approximé par une loi de distribution normale, centrée réduite, $\mathcal{N}(0,1)$. On obtient alors :

$$\text{Prob}(B_{cn} < c_1 n) = \text{Prob}(B_{cn} > (c - c_1)n) = \text{Prob}(N_{01} > \frac{c - 2c_1}{c}\sqrt{n})$$

En définissant $h = \frac{c - 2c_1}{c}\sqrt{n}$,

$$\text{Prob}\ (N_{01} > h) = \int_h^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \leq \int_{x=h}^\infty \frac{1}{\sqrt{2\pi}} e^{-\sqrt{2\pi}} d(\frac{x^2}{2}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(c-2c_1)^2}{2c} n}$$

En définissant $k_1 = \frac{2}{\sqrt{2\pi}}$ et $\ell_1 = \frac{(c-2c_1)^2}{2c}$ on obtient a).

b) Il existe des constantes positives k_2, ℓ_2 et c_2 telles que la probabilité pour qu'un ensemble de $c_1 n$ relations du type (3) et (4) ne contiennent pas $c_2 n$ relations du type (3), ainsi que $c_2 n$ relations du type (4) n'excède pas $k_2 e^{-\ell_2 n}$. La démonstration est identique à celle de a).

c) Pour tout $\varepsilon > 0$ il existe des constantes positives k_3, ℓ_3 et c_1 telles que la probabilité pour qu'un sous-graphe $G_X = (X, U_X)$ de $G = (X \cup \overline{X}, U)$, où U_X définit l'ensemble des arcs de G correspondant aux $c_1 n$ relations du type (1) et (2) choisies au hasard, ne contienne pas une composante fortement connexe avec au moins $(1 - 2\varepsilon)$ n sommets n'excède pas $k_3 e^{-\ell_3 n}$. Ceci est une conséquence directe du théorème 2 de Karp et Tarjan [11].

d) Soit X_o un ensemble de $(1 - 2\varepsilon)n$ sommets de X choisis au hasard et $\overline{X}_o$ l'ensemble correspondant dans $\overline{X}$. Il existe des constantes positives k_4 et ℓ_4 telles que la probabilité pour qu'un ensemble de $c_2 n$ relations du type (3) choisies au hasard ne contienne pas un arc joignant un sommet de X_o à un sommet de $\overline{X}_o$ n'excède pas $k_4 e^{-\ell_4 n}$. En effet cette probabilité est

est plus petite que $(2 \times \frac{2\varepsilon n}{n})^{c_2 n} \leq (\frac{1}{e})^{c_2 n}$ pour ε suffisamment petit ($\varepsilon < \frac{1}{4e}$), en prenant $k_4 = 1$ et $\ell_4 = c_2$ on obtient alors d).

e) De façon identique, il existe des constantes positives k_5 et ℓ_5 telles que la probabilité pour qu'un ensemble de $c_2 n$ relations du type (4) ne contienne pas un arc joignant un sommet de $\overline{X}_o$ à un sommet de X_o n'excède pas $k_5 e^{-\ell_5 n}$.

f) A partir de a) - e), la probabilité pour que $G = (X \cup \overline{X}, U)$ contienne un circuit impliquant une contradiction et consistant en un arc joignant un sommet de X_o (ensemble des sommets de la plus grande composante fortement connexe dans $G_X = (X, U_X)$) à un sommet de $\overline{X}_o$, un chemin dans le sous-graphe engendré par $\overline{X}_o$, un arc joignant un sommet de $\overline{X}_o$ à un sommet de X_o et un chemin dans le sous-graphe engendré par X_o n'est pas plus petite que

$$\prod_{i=1}^{5} (1 - k_i e^{-\ell_i n}) \geq 1 - \sum_{i=1}^{5} k_i e^{-\ell_i n} \geq 1 - k_6 e^{-\ell_6 n}$$

avec $k_6 = 5 \max_i k_i$ et $\ell_i = 5 \min_i \ell_i$.

Le nombre de calculs espéré peut maintenant être évalué ; l'étape 1 nécessite au pire $\Theta(n + 2n + \ldots + cn) \equiv \Theta(n)$ opérations, les étapes 2 à 4 sont en $O(n^3)$. Aussi le nombre espéré d'opérations est en $(1 - k_6 e^{-\ell_6 n})\, \Theta(n) + k_6 e^{-\ell_6 n}\, \Theta(n^3) \equiv O(n)$ ∎

4 - EXPERIENCES NUMERIQUES

L'algorithme LCA a été programmé en Fortran 77 et expérimenté sur un DPS8 (BULL). Les relations binaires sont tirées au hasard selon une loi uniforme parmi l'ensemble de toutes les relations binaires possibles sur X. Les résultats obtenus pour une série d'expériences pour n variant de 50 à 2000 et $m = \frac{n}{2}, n, \ldots, \frac{5}{2} n$ sont présentés dans le tableau 1. N_f définit le pourcentage de problèmes pour lesquels on n'obtient pas de contradiction ; N_f est égal à 100 % lorsque $m = \frac{n}{2}$ et tend également vers 100 % lorsque m = n pour les problèmes de grande taille. N_c définit le poucentage de problèmes pour lesquels on obtient une contradiction avec les ℓ premières relations binaires avec $\ell = \frac{n}{2}, n, \ldots$. Tandis que pour les problèmes de petite taille, il y a quelques différences d'un problème à un autre quant au nombre ℓ de relations binaires qui engendrent une contradiction, pour les problèmes de grande taille le nombre ℓ de relations tend vers $\frac{3n}{2}$. Le temps de calcul moyen t_{LCA}, (temps moyen obtenu sur des séries de 100 problèmes pour chacune des valeurs m et n) exprimé en secondes CPU, est donné dans la dernière colonne. Les problèmes correspondant à un nombre d'arcs m plus faible par rapport à n ($m = \frac{n}{2}$ ou m = n) utilisent les étapes 1 à 4 de l'algorithme. Pour ces problèmes, t_{LCA} augmente approximativement avec n. Les autres problèmes, où le rapport m/n est plus élevé, correspondent au cas où seule l'étape 1 est utilisé dans l'algorithme : t_{LCA} augmente alors approximativement avec n.

Des expériences de calculs ont été réalisées pour comparer l'algorithme LCA et l'algorithme de la cascade [9]. Les temps de calcul obtenus avec les mêmes données sont présentés dans le tableau 2. Les temps de calcul sont considérablement inférieurs avec l'algorithme LCA.

n	m	N_f	N_c					t_{LCA}
			$n/2$	n	$3/2n$	$2n$	$5/2n$	
50	25	100	0					0.084
	50	97	0	3				0.164
	75	50	0	2	48			0.228
	100	5	0	3	65	27		0.085
	125	0	0	4	48	45	3	0.088
100	50	100	0					0.213
	100	97	0	3				0.430
	150	27	0	3	70			0.641
	200	0	0	5	65	30		0.191
	250	0	0	5	65	30	0	0.104
200	100	100	0					0.612
	200	92	0	8				1.328
	300	9	0	4	87			0.300
	400	0	0	3	91	6		0.301
	500	0	0	4	86	0	0	0.447
300	150	100	0					1.202
	300	98	0	2				2.417
	450	0	0	4	96			0.562
	600	0	0	3	94	3		0.448
	750	0	0	5	95	0	0	0.451
400	200	100	0					1.985
	400	92	0	8				1.989
	600	0	0	7	93			0.576
	800	0	0	12	88	0		0.572
	1000	0	0	8	89	3	0	0.613
500	250	100	0					2.937
	500	87	0	13				1.207
	750	0	0	9	91			0.721
	1000	0	0	5	95	0		0.711
	1250	0	0	3	96	1	0	0.734
1000	500	100	0					12.182
	1000	98	0	2				18.653
	1500	0	0	6	94			1.258
	2000	0	0	4	96	0		1.263
	2500	0	0	3	96	1	0	1.271
2000	1000	100	0					39.6
	2000	100	0	0				69.3
	3000	0	0	3	97			2.912
	4000	0	0	0	100	0		2.715
	5000	0	0	1	99	0	0	2.834

Tableau 1 . Résolution de problèmes testsavec l'algorithme de conclusions logiques(LCA)

n = *nombre de variables*
m = *nombre de relations binaires (moitié du nombre d'arcs du graphe G)*
N_f = *pourcentage de problèmes pour lesquels aucune contradiction n'intervient*
N_c = *pourcentage de problèmes pour lesquels une contradiction est impliquée par les k premières relations, $k = n/2, n, \ldots, 5/2\, n$*
t_{LCA} = *temps de calcul de l'algorithme LCA en secondes CPU (temps moyen obtenu sur 100 problèmes).*

n	m	t_{LCA}	t_{CA}
50	25	0.084	9.8
	50	0.164	
	75	0.228	
	100	0.085	
	125	0.088	
100	50	0.213	79.1
	100	0.430	
	150	0.641	
	200	0.191	
	250	0.104	
200	100	0.612	648
	200	1.328	
	300	0.300	
	400	0.301	
	500	0.447	
300	150	1.202	2182
	300	2.417	
	450	0.562	
	600	0.448	
	750		
400	200	1.985	5178
	400	1.989	
	600	0.576	
	800	0.572	
	1000	0.613	
500	250	2.937	9938
	500	1.207	
	750	0.721	
	1000	0.711	
	1250	0.734	

Tableau 2. Comparaison des temps de calcul entre les algorithmes LCA et CA

n = nombre de variables
m = nombre de relations binaires (moitié du nombre d'arcs du graphe G)
t_{LCA} *= temps de calcul de l'algorithme LCA en secondes CPU (temps moyen sur 100 problèmes)*
t_{CA} *= temps de calcul de l'algorithme CA en secondes CPU (temps moyen sur 100 problèmes).*

REFERENCES

[1] B. APSVALL, M.F. PLASS and R.E. TARJAN : "A Linear-time Algorithm for Testing The Truth of Certain Quantified Boolean Formulas", *Information Processing Letters* 8(1979), 121-123.

[2] R. BREU and C. BURDET : "Branch and Bound Experiments in Zero-one Programming" *Mathematical Programming Study* 2 (1974) 1-50.

[3] D. COPPERSMITH and S. WINOGRAD : "On the Asymptotic Complexity of Matrix Multiplication", *SIAM Journal on Computing* 11 (1982) 472-492.

[4] H. CROWDER, E.L. JOHNSON and M. PADBERG : "Solving Large-scale Zero-one Linear Programming Problems", *Operations Research* 31 (1983) 803-834.

[5] M.J. FISHER and A.R. MEYER : "Boolean Matrix Multiplication and Transitive Closure", *Proc. 12th Annual Symposium on Switching and Automata Theory* (1971) 129-131.

[6] M. GUIGNARD and K. SPIELBERG : "Logical Reduction Methods in Zero-one Programming - Minimal Preferred Inequalities", *Operations Research* 29 (1981) 49-74.

[7] P.L. HAMMER and P. HANSEN : "Logical Relations in Quadratic 0-1 Programming", *Revue Roumaine de Mathématiques Pures et Appliquées* 26 (1981) 421-429.

[8] P.L. HAMMER and S. NGUYEN : "A Partial Order in the Solution Space of Bivalent Programs", 93-106 in N. Christofides et al. (editors) *Combinatorial Optimization*, John Wiley and Sons (1979).

[9] P. HANSEN : "A Cascade Algorithm for the Logical Closure of a Set of Binary Relations", *Information Processing Letters* 5 (1976) 50-55.

[10] B. JAUMARD and M. MINOUX : "Un Algorithme Efficace pour la Recherche de la Fermeture Transitive d'un Graphe", Note Interne CNET/PAA/TIM/MTI/1421 (Octobre 1984). Submitted for publication.

[11] E.L. JOHNSON and M.W. PADBERG : "Degree Two Inequalities, Clique Facets and Biperfect Graphs", *Annals of Discrete Mathematics* 16 (1982) 169-187.

[12] R. KARP and R.E. TARJAN : "Linear Expected-time Algorithms for Connectivity Problems", *Journal of Algorithms* 1 (1980) 374-393.

[13] I. MUNRO : "Efficient Determination of the Transitive Closure of a Directed Graph", *Information Processing Letters* 1 (1979) 56-58.

[14] P. PURDOM : "A Transitive Closure Algorithm", *BIT* 10 (1970) 76-94.

[15] B. ROY : "Transitivité et Connexité", *Compte-rendu de l'Académie des Sciences de Paris* 249 (1959) 216-218.

[16] V. STRASSEN : "Gaussian Elimination is not Optimal", *Numerische Mathematick* 12 (1969) 354-356.

[17] R.E. TARJAN : "Depth-first Search and Linear Graph Algorithms", *SIAM Journal on Computing* 1 (1972) 146-160.

[18] S. WARSHALL : "A Theorem on Boolean Matrices", *Journal of the Association for Computing Machinery* 9 (1962) 11-12.

ANNEXES

ANNEXE I

Un exemple

$X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

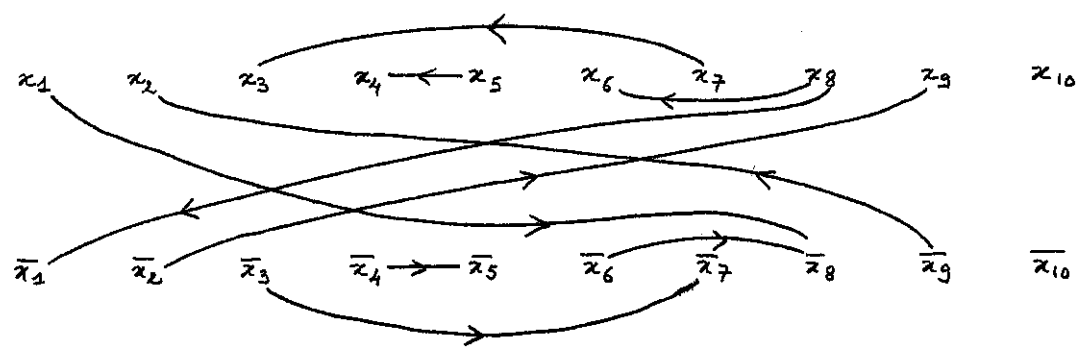
Soit R un ensemble de 32 relations binaires obtenues par un tirage selon une loi binomiale et tel que l'on ait une égale probabilité parmi les relations binaires (1) - (4).

$$R = \{x_1x_3 = 0,\ x_1x_7 = 0,\ x_1x_8 = 0,\ x_2x_7 = 0,\ x_3x_5 = 0,\ x_4x_{10} = 0,\ x_5x_7 = 0,\ x_8x_9 = 0$$
$$x_2\bar{x}_1 = 0,\ x_2\bar{x}_8 = 0,\ x_3\bar{x}_2 = 0,\ x_3\bar{x}_{10} = 0,\ x_5\bar{x}_3 = 0,\ x_5\bar{x}_4 = 0,\ x_5\bar{x}_7 = 0,\ x_5\bar{x}_9 = 0$$
$$x_3\bar{x}_5 = 0,\ x_7\bar{x}_3 = 0,\ x_7\bar{x}_5 = 0,\ x_7\bar{x}_8 = 0,\ x_8\bar{x}_2 = 0,\ x_8\bar{x}_6 = 0,\ x_6\bar{x}_3 = 0,\ x_9\bar{x}_2 = 0$$
$$\bar{x}_1\bar{x}_2 = 0,\ \bar{x}_1\bar{x}_6 = 0,\ \bar{x}_2\bar{x}_9 = 0,\ \bar{x}_2\bar{x}_7 = 0,\ \bar{x}_3\bar{x}_8 = 0,\ \bar{x}_5\bar{x}_9 = 0,\ \bar{x}_5\bar{x}_{10} = 0,\ \bar{x}_6\bar{x}_5 = 0\}$$

Soit R_o un ensemble de 5 relations binaires choisies au hasard dans R (dont les éléments ont été répartis aléatoirement) :

$$R_o = \{\ x_1x_8 = 0,\ x_5\bar{x}_4 = 0,\ x_7\bar{x}_3 = 0,\ x_8\bar{x}_6 = 0,\ \bar{x}_2\bar{x}_9 = 0\ \}$$

Soit G le graphe d'implication associé à R_o :

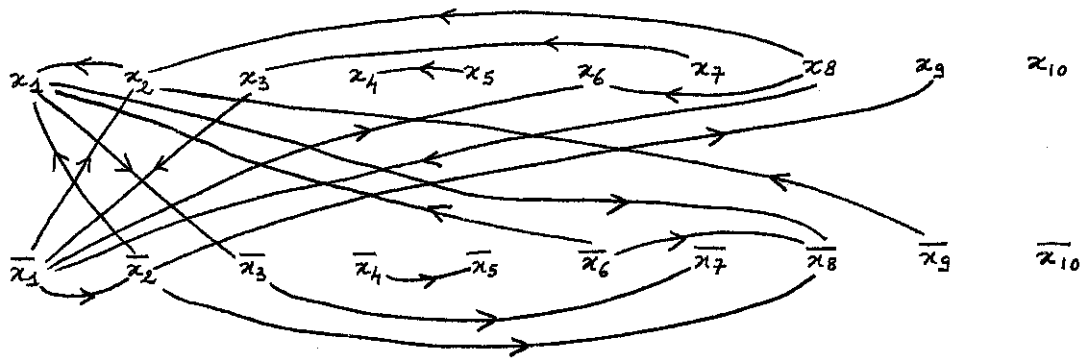


L'algorithme de recherche des composantes fortement connexes de Tarjan appliqué à G ne donne pas de contradiction.

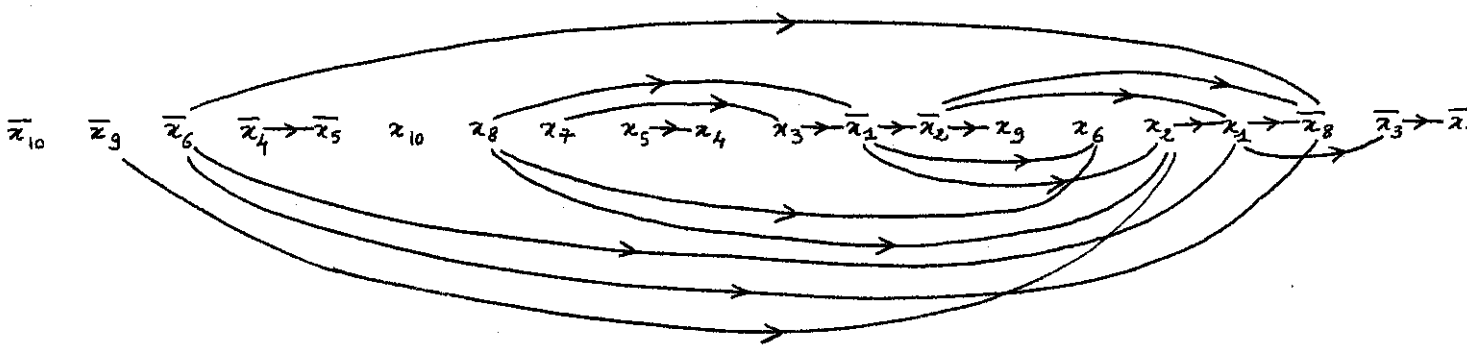
On rajoute 5 relations binaires choisies au hasard dans $R-R_o$, on obtient R_1 :

$$R_1 = R_o \cup \{x_1x_3 = 0,\ x_2\bar{x}_1 = 0,\ \bar{x}_1\bar{x}_6 = 0,\ \bar{x}_1\bar{x}_2 = 0,\ x_8\bar{x}_2 = 0\}$$

Soit G le graphe d'implication associé à R_1 :



Chaque sommet, représentant sur cet exemple, à cette étape, une composante fortement connexe, le graphe réduit G_{R_1} déduit de G_1 est :



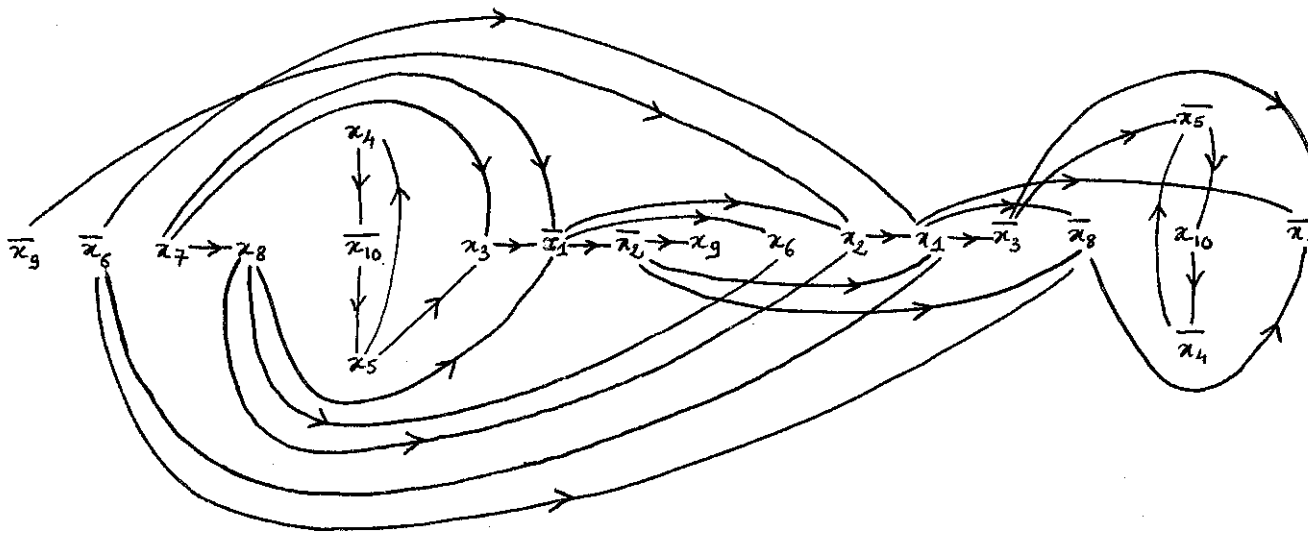
Si l'ensemble des relations binaires considéré était égal à R_1, on pourrait conclure qu'il existe au moins une solution (pas de contradiction dans G_{R_1}). De plus, il existe un chemin de $\bar{x}_1$ à x_2 et un chemin de $\bar{x}_1$ à $\bar{x}_2$ (traduction des relations binaires $\bar{x}_1\bar{x}_2 = 0$ et $\bar{x}_1x_2 = 0$), ce qui permet de déduire que $x_1 = 1$. De même, il existe un chemin de x_8 à $\bar{x}_1$ et un chemin de x_8 à x_1 dont on déduit que $x_8 = 0$

(ce que l'on aurait également pu obtenir en remarquant qu'il existait un chemin de x_1 à $\bar{x}_8$, soit que $x_1 = 1 \leq \bar{x}_8$, soit $\bar{x}_8 = 1$). Par un raisonnement analogue, on obtient également $x_3 = 0$ et $x_7 = 0$.

Revenons à l'ensemble R de départ. Le graphe G_{R_1} n'impliquant pas de contradiction, rajoutons 5 relations binaires (soit 10 arcs) choisies au hasard dans $R - R_1$.

On obtient $R_2 = R_1 \cup \{\bar{x}_5\bar{x}_{10} = 0,\ x_1x_7 = 0,\ x_5\bar{x}_3 = 0,\ x_{10}x_4 = 0,\ x_7\bar{x}_8 = 0\}$

Soit G_2 le graphe d'implication associé à R_2.



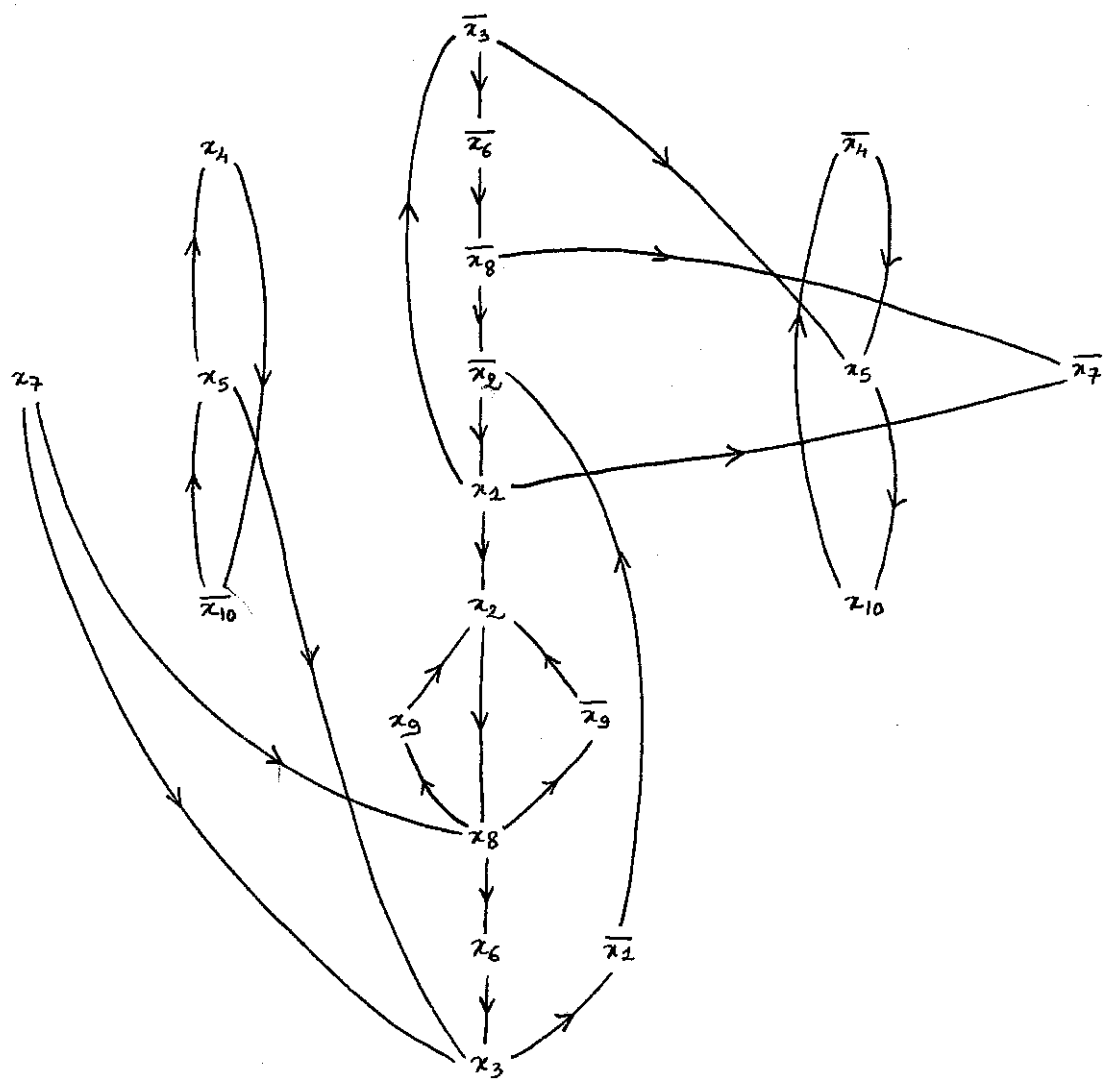
G_{R_2} comporte 16 composantes fortement connexes qui n'impliquent pas de contradiction. On peut déduire de G_{R_2} les conclusions logiques suivantes :

- identification de variables : $x_4 = x_5 = \bar{x}_{10}$
- fixation de variables : $x_1 = 1,\ x_3 = 0,\ x_4 = 0,\ x_5 = 0,\ x_8 = 0,\ x_{10} = 1.$

Rajoutons encore 5 relations binaires choisies au hasard dans $R - R_2$. On obtient R_3 :

$$R_3 = R_2 \cup \{x_9\bar{x}_2 = 0,\ x_2\bar{x}_8 = 0,\ x_6\bar{x}_3 = 0,\ \bar{x}_3\bar{x}_8 = 0,\ x_8x_9 = 0\}$$

Soit G_3 le graphe d'implication associé à R_3 (déduit de l'algorithme de Tarjan) :



(Tous les arcs n'ont pas été représentés)

x_1 et $\bar{x}_1$ appartiennent à la même composante fortement connexe, on a alors une contradiction : on peut arrêter l'algorithme, il n'est plus possible de trouver une solution.

ANNEXE II

Programme de l'algorithme CA

```
c n est le nombre de sommets du graphe d'implication, n=2nv
c nv est le nombre de variables
           parameter(nv=50,n=2*nv)
           common/limite/mliml,mlim2
           integer xi,xj
           common/tab/xi(50000),xj(50000)
           integer mul(16,16),som(16,16)
           integer m
           common/mat/m(500,500)
c
c
c lecture de la table definissant l'addition des etats
           open(11,file='addition',form="formatted")
           do 10 i=1,16
           read(11,900) (som(i,k),k=1,16)
    10     continue
c
c lecture de la table definissant la multiplication des etats
           open(12,file='multiplication',form="formatted")
           do 20 i=1,16
           read(12,900) (mul(i,k),k=1,16)
     20    continue
c
          tcpu=0.
          write(06,901) nv,2*nv
c mliml est la valeur maximale du nombre d'arcs generes
c relativement aux dimensionnements des tableaux
          mliml=50000
          do 100 ktirage=1,100
c
          write(06,902) ktirage
c
c tirage selon une loi binomiale de parametre prob
          prob=0.1
          print,"parametre loi binomiale",prob
          call random(n,nv,nr,prob)
          print,'graphe initial de',nr,'arcs'
c
c construction de la demi-matrice des etats initiaux
           do 40 i=1,nv
           do 30 k=1,nv
           m(i,k)=1
    30     continue
    40     continue
           do 60 j=1,nr
           ii=xi(j)
           kk=xj(j)
           if(ii.gt.nv) then;i=ii-nv;
                        else;i=ii;ii=ii+nv;
           endif
           if(kk.gt.nv) then;k=kk-nv;
                        else;k=kk;kk=kk+nv;
           endif
           if((i.eq.xi(j)).and.(k.eq.xj(j))) then;mm=3;goto 50;endif
           if((i.eq.xi(j)).and.(kk.eq.xj(j))) then;mm=2;goto 50;endif
           if((ii.eq.xi(j)).and.(kk.eq.xj(j))) then;mm=4;goto 50;endif
           if((ii.eq.xi(j)).and.(k.eq.xj(j))) then;mm=5;goto 50;endif
     50    m(i,k)=som(mm,m(i,k))
     60    continue
c
```

```
c
c calcul de la fermeture transitive
          call ptime(t0)
          do 90 j=1,nv
          do 80 i=1,nv
          do 70 k=1,nv
          m(i,k)=som(m(i,k),mul(m(i,j),m(j,k)))
     70   continue
     80   continue
     90   continue
          call ptime(t1)
c
          tcpu=tcpu+t1-t0
          write(06,903) (tcpu*3600.)/float(ktirage)
c
    900   format(16i3)
    901   format(i5,13h variables ou,i5,8h sommets)
    902   format(/,7h tirage,i4)
    903    format(24h temps d'execution moyen,e14.6,13h secondes cpu)
c
    100    continue
          stop
          end
```

ANNEXE III

Programme de l'algorithme LCA

```
c nv est le nombre de variables
c n est le nombre de sommets du graphe, n=2nv
      parameter(nv=50,n=2*nv)
c
      common/limite/mlim1,mlim2
      integer xi,xj
      common/tab/xi(50000),xj(50000)
      integer pteur,arc,vert
      common/gi/pteur(60000),arc(60000),vert(5000)
      integer comp,rep,elt
      common/cofc/comp(5001),rep(5000),elt(5000)
      integer somr,sucr
      common/gr/somr(5001),sucr(50000)
      integer pf,s
      common/ftrgr/pf(5001),s(100000)
      integer som,suc
      common/ft/som(5001),suc(100000)
      integer pas1,pas2,zz,x,y
      logical ecrit,trouve
c
c si ecrit=vrai alors impression des resultats intermediaires
      ecrit=.false.
      write(06,900) nv
c tirage selon une loi binomiale de parametre prob
      prob=0.1
      write(06,901) prob
c
c pour les dimensionnements de tableaux
c mlim1 est la valeur maximale du nombre d'arcs dans le graphe initial
      mlim1=50000
c mlim2 est la valeur maximale du nombre d'arcs dans
c la fermeture transitive du graphe reduit
      mlim2=100000
      tcpu=0.
c
      do 100 ktirage=1,100
      write(06,902) ktirage
c
c generation aleatoire de paquets d'arcs
      call random(n,nv,nr,prob)
c
      write(06,903) nr
      nr1=nr
      nr=3*nv
      if(nr.gt.nr1)then;print,'pas assez d"arcs generes';
                        goto 100;
                        endif
      write(06,904) nr,nr/2
c
c initialisation de la structure de donnees
      do 10 i=1,mlim1+n
      pteur(i)=0
      arc(i)=0
 10   continue
      do 20 i=1,n
      vert(i)=0
 20   continue
      narc=n
c
c si trouve=false alors on a trouve une contradiction en
```

```
c determinant les composantes fortement connexes
          trouve=.true.
c
c on prend un paquet de pas2-pas1 arcs
c
c initialisation
          pas1=1
          pas2=nv
          if(nr.lt.nv) pas2=nr
c
c remplissage de la structure de donnees qui
c contient le graphe
c
c nk=nombre d'arcs generes non encore utilises
          nk=nr
c
  30      call ptime(tp0)
          do 40 i=pas1,pas2
          nk=nk-1
          x=xi(i)
          y=xj(i)
          if(vert(x).eq.0) then;vert(x)=x;
                                pteur(x)=-1;
                                arc(x)=y;
                           else;zz=vert(x);
                                narc=narc+1;
                                pteur(zz)=narc;
                                pteur(narc)=-1;
                                arc(narc)=y;
                                vert(x)=narc;
          endif
  40      continue
          call ptime(tp1)
c
c calcul des composantes fortement connexes
          if(pas1.eq.1)then;call ptime(tpi);
c on ne prend pas en compte le chargement de la structure de donnees
c pour le temps cpu
                          else;tpi=tpi-(tp1-tp0);
          endif
          call cfc(n,nv,ncfc,trouve)
          if(ecrit) call impr1(n,ncfc,nr)
c
c si on a trouve une contradiction (trouve=false)
c alors FIN : on va en 90
          if(.not.trouve) goto 90
c
c si pas de contradiction, on ajoute un
c nouveau paquet d'arcs
          if(nk.ge.nv) then;pas1=pas2+1;
                            pas2=pas2+nv;
                            goto 30;
                       else;if(nk.ne.0)then;pas1=pas2+1;
                                           pas2=nr;
                                           goto 30;
                            endif;
          endif
c
c
c il n'y a pas de contradiction :
```

```
c recherche de la fermeture transitive des relations binaires
c
c 1 ere etape : determination du graphe reduit
          call reduc(n,ncfc)
c
c 2 eme etape : fermeture transitive du graphe reduit
          call ftrans(n,ncfc)
c
c 3 eme etape : fixation de variables
          call fixe(nfix,n,nv)
c
  90      call ptime(tpf)
          tcpu=tcpu+(tpf-tpi)
          write(06,905) (tpf-tpi)*3600.
          write(06,906) ktirage,(tcpu*3600.)/float(ktirage)
          if(.not.trouve)then;print,'CONTRADICTION';goto 100;
                         else;write(06,907) nfix;
          endif
          if(ecrit) call impr2(n,ncfc)
c
  100     continue
c
  900     format(3x,i4,10h variables,/)
  901     format(30h parametre de la loi binomiale,2x,e12.6,/)
  902     format(/,7h tirage,i4)
  903     format(i5,42h relations binaires generees aleatoirement)
  904     format(7h graphe,/,i5,8h arcs ou,i5,19h relations binaires)
  905     format(28h duree d'execution du tirage,e14.6,13h secondes cpu)
  906     format(19h duree moyenne pour,i4,10h tirages :,e14.6,13h secondes cpu
)
  907     format(i5,17h variables fixees)
          stop
          end
c
c
c
c
c
c tirage d'un ensemble d'arcs selon une loi
c binomiale de parametre prob
          subroutine random(n,nv,nr,prob)
c
          double precision g05caf
          integer z,top,yi,yj
          integer xi,xj
          common/tab/xi(50000),xj(50000)
          common/limite/mlim1,mlim2
c
c g05caf returns a pseudo-random number taken from a
c uniform distribution between 0 and 1
c
c g05ccf sets the basic generator routine g05caf to a
c non-repeatable initial state
          call g05ccf
c
c generation de nr relations binaires
c selon une loi binomiale
          nr=0
c
c relations du type xi.(1-xj) ou xi<=xj
```

```
          do 20 i=1,nv-1
          do 10 j=i+1,nv
          r=sngl(g05caf(1.0d0))
          if(r.ge.prob) goto 10
          nr=nr+1
          xi(nr)=i
          xj(nr)=j
   10     continue
   20     continue
c
c relations du type (1-xi).xj=0 ou 1-xi<=1-xj
          do 40 i=nv+1,n-1
          do 30 j=i+1,n
          r=sngl(g05caf(1.0d0))
          if(r.ge.prob) goto 30
          nr=nr+1
          xi(nr)=i
          xj(nr)=j
   30     continue
   40     continue
c
c relations du type xi.xj=0 ou xi<=1-xj
          do 60 i=1,nv-1
          do 50 j=i+nv+1,n
          r=sngl(g05caf(1.0d0))
          if(r.ge.prob) goto 50
          nr=nr+1
          xi(nr)=i
          xj(nr)=j
   50     continue
   60     continue
c
c relations du type (1-xi).(1-xj)=0 ou 1-xi<=xj
          do 80 i=nv+1,n-1
          do 70 j=i-nv+1,nv
          r=sngl(g05caf(1.0d0))
          if(r.ge.prob) goto 70
          nr=nr+1
          xi(nr)=i
          xj(nr)=j
     70   continue
     80   continue
c
          if(2*nr.gt.mliml) then;print,"erreur dimension de tableau";stop;endif
c
c randomisation des relations binaires
          top=nr
          do 100 i=1,nr
          zalea=sngl(g05caf(1.0d0))*(top-0.5)+1.
          z=int(zalea)
          yi=xi(z)
          yj=xj(z)
          do 90 j=z,top
          xi(j)=xi(j+1)
          xj(j)=xj(j+1)
   90     continue
          xi(top)=yi
          xj(top)=yj
          top=top-1
   100    continue
```

```
c
c rangement aleatoire par paquets de nv relations
          top=nr
          do 120 i=1,nr
          zalea=sngl(g05caf(1.0d0))*(top-0.5)+1.
          z=int(zalea)
          yi=xi(z)
          yj=xj(z)
          do 110 j=z,top
          xi(j)=xi(j+1)
          xj(j)=xj(j+1)
  110     continue
          xi(2*top)=yi
          xj(2*top)=yj
          if(yj.gt.nv)then;xi(2*top-1)=yj-nv;else;xi(2*top-1)=yj+nv;endif
          if(yi.gt.nv)then;xj(2*top-1)=yi-nv;else;xj(2*top-1)=yi+nv;endif
          top=top-1
  120     continue
c
          return
          end
c
c
c
c
c
c calcul des composantes fortement connexes
c algorithme de Tarjan
        subroutine cfc(n,nv,ncfc,trouve)
c
        integer num(5000),p(5000),ns(5000),inf(5000)
        integer stack(5000),mark(5000),point(5000)
        integer v,w,top
        integer pteur,arc,vert
        common/gi/pteur(60000),arc(60000),vert(5000)
        integer comp,rep,elt
        common/cofc/comp(5001),rep(5000),elt(5000)
        logical trouve
c
c initialisation des tableaux et des parametres
        do 10 j=1,n
        rep(j)=0
        num(j)=0
        p(j)=0
        if(pteur(j).eq.0) then;ns(j)=-1;else;ns(j)=j;endif
        inf(j)=0
c comp(j)=numero de la comp. fort. connexe du sommet j
        comp(j)=0
        elt(j)=0
        stack(j)=0
c mark(j)=1 ssi le sommet j appartient deja a une comp. fort. connexe
        mark(j)=0
        point(j)=0
 10     continue
        i=1
        comp(1)=1
        comp(n+1)=0
c ncfc est le nombre de composantes fortement connexes
        ncfc=0
c top pointe sur le sommet de la pile (tableau stack)
```

```
        top=0
        ii=1
c nmark est le nbre de sommets non encore marques
        nmark=n
c
c algorithme
c
c choix d'un sommet v non encore marque
 20     if (mark(ii).eq.0) goto 30
        ii=ii+1
        goto 20
 30     v=ii
c
c initialisation pour le sommet racine de l"arborescence
        inf(v)=i
        num(v)=i
c v est marque
        mark(v)=1
        nmark=nmark-1
c on met v sur le sommet de la pile
        top=top+1
        stack(top)=v
c point(v)=1 ssi v est dans la pile
        point(v)=1
c
c construction de l"arbre
c goto 200 si tous les successeurs de v ont deja ete examines
 40     if(ns(v).eq.(-1)) goto 70
c w est un successeur de v
        w=arc(ns(v))
        ns(v)=pteur(ns(v))
c
        if(mark(w).ne.0) goto 50
        i=i+1
        nmark=nmark-1
        num(w)=i
        inf(w)=i
        p(w)=v
        mark(w)=1
        top=top+1
        stack(top)=w
        point(w)=1
        v=w
        goto 40
 50     if((num(w).ge.num(v)).or.(point(w).eq.0)) goto 60
        inf(v)=min0(inf(v),num(w))
 60     goto 40
 70     if(num(v).ne.inf(v)) goto 110
c v nouvelle racine d"une comp. fort. connexe
c ncfc est le numero de la comp. fort. connexe
        ncfc=ncfc+1
        kk=comp(ncfc)
c recherche des sommets qui appartiennent a la meme cfc que xi
c on depile jusqu"a v compris
        w=stack(top)
 80     if(num(w).lt.num(v)) goto 90
        elt(kk)=w
        rep(w)=ncfc
c w est retire de la pile
        point(w)=0
```

```
          endif
          jj=suc(j)
          if(jj.gt.nv) jj=jj-nv
          if(c(jj).eq.1) then;vfix(i)=0;
                              vfix(i+nv)=1;
                              nfix=nfix+1;
                              goto 90;
                         else;c(jj)=1;
          endif
    60    continue
          goto 90
c
c fixation de la variable xi a un
  70      do 80 j=1,11
          if(suc(j).eq.(i-nv)) then;vfix(i-nv)=1;
                                    vfix(i)=0;
                                    nfix=nfix+1;
                                    goto 90;
          endif
          jj=suc(j)
          if(jj.gt.nv) jj=jj-nv
          if(c(jj).eq.1) then;vfix(i-nv)=1;
                              vfix(i)=0;
                              nfix=nfix+1;
                              goto 90;
                         else;c(jj)=1;
          endif
   80     continue
c
   90     continue
c
          return
          end
c
c
c
c
c
c impression du graphe d'implication et des composantes fortement connexes
           subroutine impr1(n,ncfc,nr)
c
           integer pteur,arc,vert
           common/gi/pteur(60000),arc(60000),vert(5000)
           integer comp,rep,elt
           common/cofc/ comp(5001),rep(5000),elt(5000)
c
           print,'graphe d"implication'
           write(06,95)nr
           do 20 i=1,n
           if(vert(i).eq.0) then;
                            write(6,93)i;
                            else;
                            write(6,90)i;
                            k=i;
  10                        if(pteur(k).ne.(-1)) then;write(6,91)arc(k);
                                                       k=pteur(k);
                                                       goto 10;
                                                  else;write(6,92)arc(k);
                            endif;
           endif
```

```
  20       continue
c
           print,'composantes fortement connexes'
           i=comp(1)
           do 30 j=1,ncfc
           ii=comp(j+1)
           if(i.eq.ii) then;write(06,93)j;
                       else;write(06,94)j,(elt(k),k=i,ii-1);
           endif
           i=ii
  30       continue
c
c
  90       format(3x,i3,8x,$)
  91       format(i4,$)
  92       format(i4)
  93       format(3x,i3,8x,19h pas de successeurs)
  94       format(3x,i3,8x,14i4,/,100(14x,14i4,/))
  95       format(i4,5h arcs,/)
c
           return
           end
c
c
c
c
c
c impression du graphe reduit et de sa fermeture transitive
         subroutine impr2(n,ncfc)
c
           integer somr,sucr
           common/gr/somr(5001),sucr(50000)
           integer pf,s
           common/ftrgr/pf(5001),s(100000)
c
           print,'graphe reduit'
           i=somr(1)
           do 10 j=1,ncfc
           ii=somr(j+1)
           if(i.eq.ii)then;write(06,93)j;
                      else;write(06,94)j,(sucr(k),k=i,ii-1);
           endif
           i=ii
  10       continue
c
           print,'fermeture transitive du graphe reduit'
           i=pf(1)
           do 100 j=1,ncfc
           ii=pf(j+1)
           if(i.eq.ii) then;write(06,93)j;
                       else;write(06,94)j,(s(k),k=i,ii-1);
           endif
           i=ii
  100      continue
c
  93       format(3x,i3,8x,19h pas de successeurs)
  94       format(3x,i3,8x,14i4,/,100(14x,14i4,/))
c
           return
           end
```