Modélisation et analyses statistiques sur des simulations des composants électroniques

BELKHEIR Abdelmadjid

Master -ICA- Spécialité *MASSS*
2005 – 2006

Stage de 3 avril 2006 au 30 septembre 2006

Remerciements

Je tiens à remercier

– Mr Julien Torrès qui m'a accompagné durant mon stage et également toute l'équipe de la société *Sertess* pour leur accueil.

– Mr Gérard D'aubigny pour ces précieux conseils.

Introduction

Ce stage rentre dans le cadre d'un master professionnel **ICA** (Information Cognition Apprentissage), spécialité **MASSS** (Modélisation et Apprentissage Statistique en Sciences Sociales) de l'Université Pierre-Mendès-France (UPMF-Grenoble).
Cette formation généraliste à la Statistique, couvre le champ de la Statistique appliquée et entend former des spécialistes du traitement de l'information et de la modélisation dans l'incertain, directement opérationnels et aptes à dialoguer tant avec des chercheurs (informaticiens, biologistes, ...), des ingénieurs, des médecins qu'avec des cadres formés aux disciplines des Sciences Sociales : psycho-sociologues, économistes,...

CERTESS est la société accueillente, une jeune société de la région grenobloise (Moirans) qui fournit des solutions logicielles pour l'industrie des semi-condicteurs (EDA - Electronic Design Automation). *Certess* propose notamment un outil (logiciel *Certitude*) qui s'intègre dans le flot test et de validation d'un composant électronique, immédiatement après sa conception. En effet, le coût d'une erreur dans le composant détectée après sa gravure étant très élevé, il est primordial que l'environnement de test soit de très bonne qualité. L'outil développé par *Certess* permet précisément de mesurer la qualité de l'environnement de test et d'en trouver les faiblesses, avant que le composant ne soit envoyé à la gravure.

Le but de ce stage est de faire des analyses statistiques sur des simulations des composants électroniques, afin de proposer des méthodes de modélisation et de prévision de la mesure de la qualité d'un environnement de test (plus en détail dans la première partie).

La première partie est une visite guidée dans le monde de l'E.D.A, et plus précisément celui de *Certess* afin de comprendre le fonctionnement de *Certitude* et la problématique posée pour ce stage. On y présentera aussi les données disponibles pour l'étude. La deuxième partie constitue une exploration et une déscription des données en utilisant des théories et des méthodes d'analyse de données multidimentionnelles appliquées notamment sur des données qualitatives. La troisième partie est réservée à la modélisation et la prévision de la mesure de la qualité d'un environnement de test par la régression logistique.

Table des matières

Première partie
Présentation et position de problème

1 Principes de développement des composants électroniques

Un *circuit électronique* est constitué de portes logiques reliées entre elles afin d'obtenir un comportement spécifique. Le développement d'un cicuit passe par plusieurs étapes qu'on appelle *le flot de développement* présenté figure 1, page 6 :

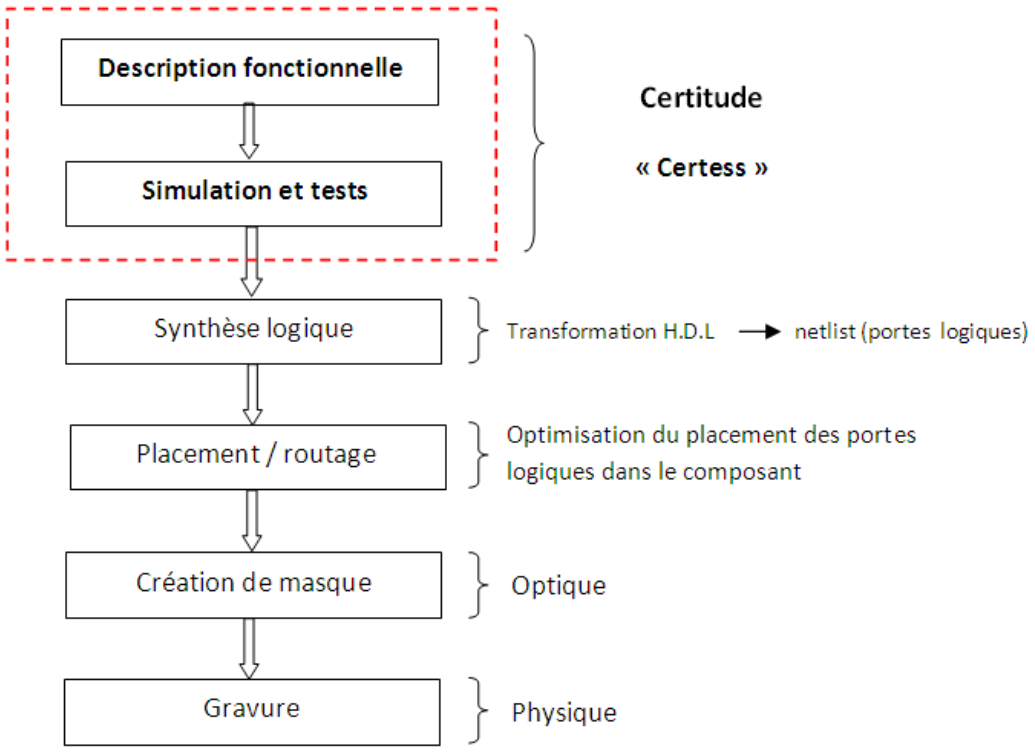

FIG. 1 – Flot de développement d'une puce électronique

La complexité de chacune des étapes du *flot de développement* rend nécessaire l'utilisation de logiciels dédiés. L'**E.D.A** (Electronic Design Au-

tomation) est l'industrie qui fournit les logiciels pour le développement des composants électroniques.

La société *Certess* développe un logiciel E.D.A appelé *Certitude*. Ce logiciel s'insère dans les deux premières étapes du flot de développement. On va donc présenter plus en détail ces deux étapes :

La *description fonctionnelle* :
On décrit tout composant électronique grâce à un langage **H.D.L** (Hardware Description Language). Les deux principaux langages utilisées dans l'industrie sont *Verilog* et *VHDL*. On appelle *design* le circuit électronique sous sa forme de description fonctionnelle en langage H.D.L. Un design est donc tout à fait similaire à un programme informatique.

Simulations et tests :
Pour pouvoir interpréter le langage H.D.L et simuler le comportement du design, nous avons besoin des *simulateurs* par exemple les programmes VCS, NCSim et ModelSim.

Les *tests* servent à la vérification du comportement du circuit (simulé) par rapport au cahier des charges. Le protocole de mise en oeuvre des tests est décrit figure 2, page 7. Le circuit (Design Under Test - D.U.T) va être soumis à divers stimuli dont on connait les résultats grâce à un modèle de référence. On compare donc les sorties du design et celles du modèle de référence soumis aux mêmes stimuli pour déterminer si le comportement du composant est celui qui est attendu.

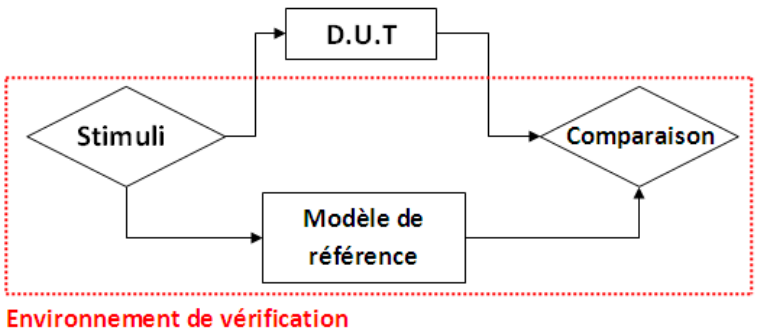

FIG. 2 – Environnement de test

2 Mesure de la qualité d'un environnement de tests

2.1 Problématique

Les problématiques posées par *Certess* sont :

- Quelle est la qualité des tests ?
- Les tests sont-ils suffisants ? Et est-ce qu'il manque des tests ?
- Est-ce que toutes les parties du circuit sont bien testées ?

Le logiciel *Certitude* est produit par *Certess* pour répondre aux questions ci-dessus

2.2 Le logiciel *Certitude*

Le logiciel *Certitude* est un outil d'analyse par mutation. Le principe de l'analyse par mutation (*Mutation Analysis*) consiste à injecter des *fautes* (bugs) au niveau H.D.L. Voir ci-dessous un exemple de fautes (figure 3, page 8) :

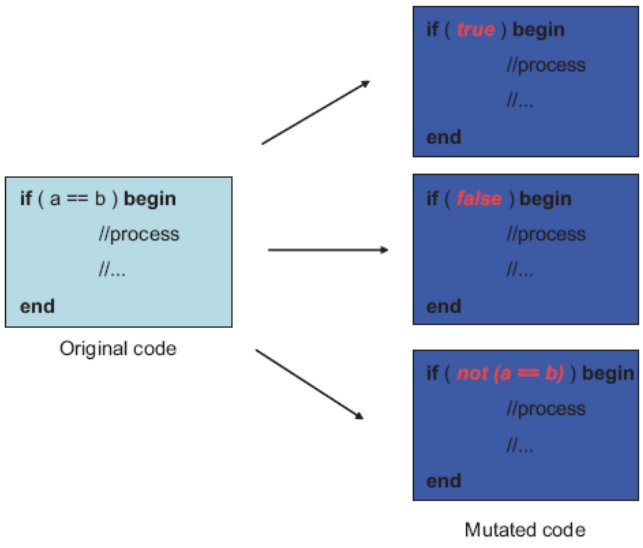

FIG. 3 – Exemple des fautes

Le code original fait intervenir la condition $a == b$. Les trois type de fautes obtenus par mutation consistent ici :

– Soit à substituer la valeur *true* au test a==b.
– Soit à substituer la valeur $false$ au test a==b.
– Soit à substituer le test $not(a == b)$ au test a==b.

Quand on injecte une faute dans le circuit, il va se comporter de façon erronée. Donc il doit exister au moins un test qui doit *détecter* ce comportement incorrect. S'il n'existe aucun test qui détecte la faute injectée, alors l'environnement de test est incomplet ou insuffisant et il doit être amélioré.

2.3 Les fonctions de Certitude

La conception du logiciel Certitude a conduit à distinguer trois phases (modélisation, activation, détection).

2.3.1 La phase de modèlisation (model phase)

La phase de modèlisation génère le code instrumenté (c'est à dire la description du design en langage H.D.L). On introduit les fautes dans le code du design (le nombre des fautes est en général de quelques milliers). Il existe dix-sept types de fautes adoptés par Certess, (on note *type-faute*) regroupées dans deux grandes catégories :

– **Control flow** : CaseItemDead, ConditionFalse, ConditionTrue, ElseDead, ForDead, ForSkipFirst, ForSkipLast, NegatedCondition, WhileDead. Ces fautes agissent directement sur les débranchements dans le design.

– **Autres fautes** : BitFlip, DeadAssign, FlipFirst, FlipLast, Operator, PrefixExpression, StringConstant, SwapOperand.

Remarque : Dans un design, on ne trouve pas forcément tous les types de fautes.

2.3.2 La phase d'activation

Au cours de cette phase, *Certitude* éxécute une fois chacun des tests sans les fautes, mais avec des « marqueurs » là où les fautes sont insérées pour établir quels sont ceux qui vont activer telle ou telle faute. Par exemple, si

une partie du circuit n'est pas utilisée par un test donné, alors les fautes insérées dans cette partie ne seront pas activées par ce test.
Cette information est très importante. Savoir quelle faute est acvtivée par quel test permet de limiter le nombre des simulations à effectuer lors de la phase de *détection*.

La comparaison effectuée en fin de test (figure 2, page 7) conduit à trois cas de figures illustrés dans l'exemple suivant (figure 4, page 10) :

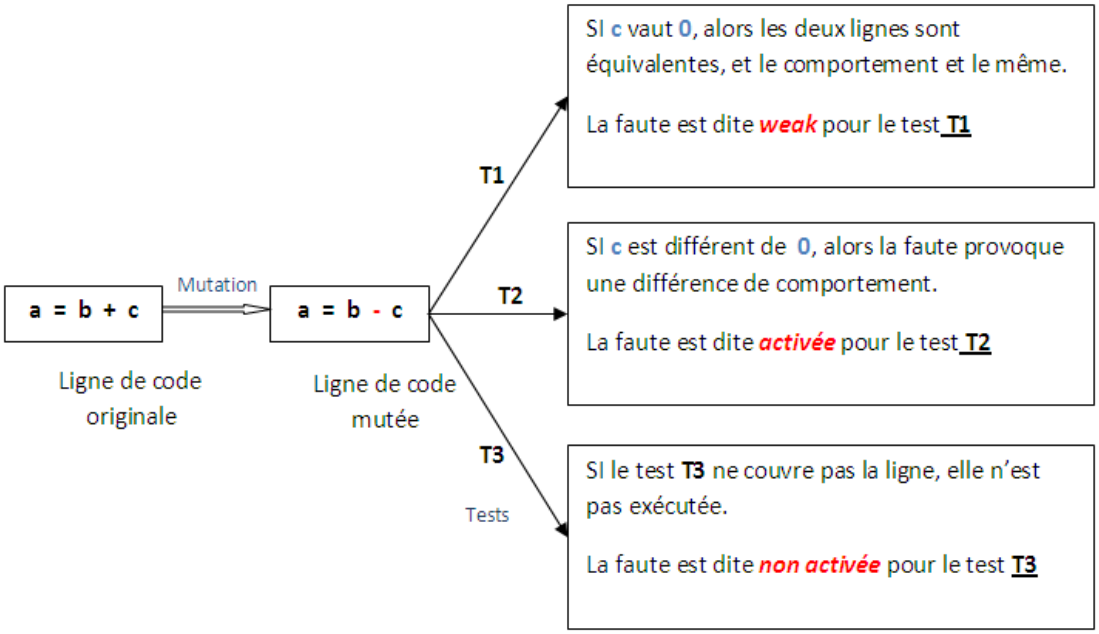

FIG. 4 – Les statuts d'une faute

Non activée : Une faute est non activée pour un test donné si le test ne couvre pas la faute : Le déroulement du programme ne passe pas par l'endroit où la faute a été insérée.

Activée : Une faute est activée pour un test donné si le test couvre la faute et la faute provoque une différence de comportement.

Weak : Une faute est *weak* pour un test donné si cette faute ne provoque pas de différence dans le contexte du test.

Statuts finaux des fautes

- Une faute activée par au moins un test a le statut *activée*.
- Une faute non activée par tous les tests a le statut *non activée*.
- Une faute non activée par certains tests et weak pour tous les autres tests a le statut *weak*.

	F_1	F_2	F_3	F_4	F_5	F_6
T_1	NA	NA	NA	NA	W	A
T_2	W	NA	W	NA	A	A
T_3	W	NA	A	W	NA	A
$statut final$	W	NA	A	W	A	A

TAB. 1 – Tableau des status finaux

W : weak, NA : non activée, A : activée.

En pratique, le nombre de fautes est superieur au nombre de tests.

2.3.3 La phase de détection

Dans cette phase, on exécute les tests avec les fautes insérées. Pour éviter les interactions, on insère les fautes une par une. Le mécanisme de détection consiste à insérer une faute et éxécuter les tests qui l'activent (un par un) jusqu'à ce qu'un test la détecte, dans ce cas la faute est *détectée*. Si aucun test ne détecte cette faute, la faute est *non détectée*.

Une faute est *propagée* par un test donné si ce test propage la différence de comportement induite par la faute injectée jusqu'au moins un port de sortie du design. C'est un statut intermédiaire, on ne lui donnera pas trop d'importance dans ce rapport.

Métrique :
A la fin de la phase de détection, on peut calculer une mesure de la fiabilité d'un environnement de tests. Une telle mesure est appelée la *métrique* du design. La *métrique* représente le pourcentage des fautes détectées parmi les fautes activées dans le *design*.

Certaines fautes ne sont pas détectables (insuffisance de l'environnement de test). Pour le montrer il faut donc exécuter tous les tests qui l'activent. Les fautes non détectables sont donc très couteuses en temps. On va donc

chercher à estimer la *métrique* (ou le nombre de fautes non détectables) en minimisant le nombre d'exécution de tests.
Ce stage se déroulait dans ce contexte, pour répondre à ce besoin :

- *Est-ce que la phase d'activation est suffisante pour prévoir la détection ?*
- *Quelles sont les critères pertinents qui discriminent entre les deux groupes de fautes (détectées et non détectées) ?*
- *Existe-t-il un moyen pour modéliser ce phénomène ?*

On peut présenter les statuts successifs d'une faute sur le schéma suivant :

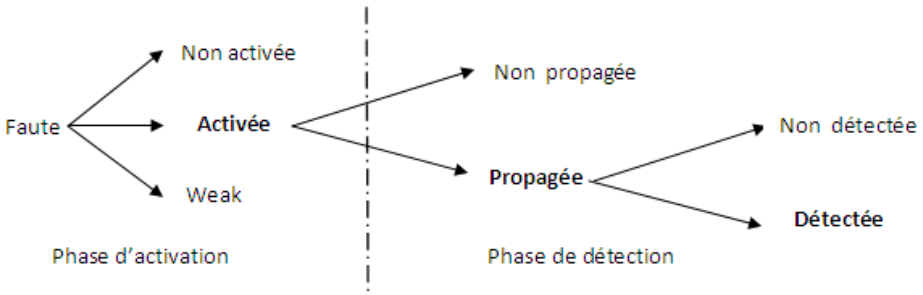

FIG. 5 – Qualification d'une faute dans un environnement de test

3 Présentation de quelques designs

Au cours de ce stage, quatre designs différents ont été mis à ma disposition pour lequels on connaissait les résultats (en activation et en détection) de tous les tests sur chacune des fautes : ces résultats avaient été préalablement enregistrés par Certess.

*Hc*11*verilog* (micro-contrôleur) :
L'étude s'est fondée essentiellement sur ce design, car il est considéré par les cadres de Certess comme le plus représentatif en terme de nombre de fautes et de nombre de tests.

*I*8051 (micro-contrôleur) :
Le *i*8051 contient peu de tests mais plus de fautes par rapport au *hc*11*verilog*.

*Router*2 **:**
C'est un prototype construit par *Certess*.

Edac :
Il est similaire au *hc*11*verilog* en terme de taille, mais il est très particulier et ne représente pas beaucoup de cas réels. Les tests sont regroupés d'une façon parfaite, où chaque groupe s'intéresse à un ensemble de fautes séparément des autres groupes.

Le tableau ci-dessous (voir 2, page 13), résume les caracteristiques des quatre designs :

	hc11verilog	i8051	router2	edac
Nombre de tests	83	9	100	100
Nombre de fautes	2523	3368	278	4240
Nombre de fautes activées	2297	3042	244	3381
Nombre de fautes weak	89	120	14	0
Nombre de fautes propagées	2163	2905	244	3160
Nombre de fautes détectées	2056	2895	189	2820
Métrique	0.8951	0.9516	0.7746	0.8341

TAB. 2 – Les caractéristiques des quatre designs

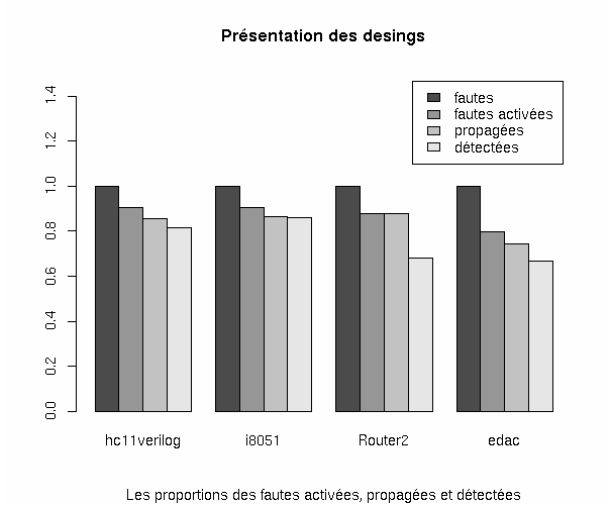

FIG. 6 – diagramme en bâton des quatre designs

4 D'autres définitions

On donne ici quelques définitions supplémentaires de notions utilisées dans le rapport.

Firsttime : C'est le temps de la première activation d'une faute F par un test T. Cette information est disponible après la phase d'activation pour toutes les fautes activées et pour tous les tests. On code « -2 » les fautes *non activées* et « -1 » les fautes *weak*. Un exemple de matrice de $firsttime$ est présenté dans la figure 3, page 14.
Remarque : A l'image de la faute 5 dans cet exemple, les fautes activées par tous les tests sont le plus souvent à temps d'activation égale à 0 (initialisation).

	Test 1	Test 2	Test 3	. . .	Test j	. . .	Test p
Faute 1	100	200	2600	. . .	1	. . .	-1
Faute 2	1300	-1	-2	. . .	-1	. . .	-1
Faute 3	5000	-2	-1	. . .	2500	. . .	360
Faute 4	-2	-1	1	. . .	-2	. . .	100
Faute 5	0	0	0	. . .	0	. . .	0
.	.	.	.	. . .	.	. . .	.
.	.	.	.	. . .	.	. . .	.

Tab. 3 – Exemple d'une matrice des $firsttime$.

Poids des fautes : on note *poids-faute*. Cette variable quantitative est calculée de la façon suivante :
Si la faute est de type *autre faute*, le poids correspond au nombre de fautes dans la ligne ; Si la faute est de type *control flow*, le poids correspond au nombre de fautes liées.

Profondeur : Il est plus simple de définir la profondeur grâce à l'exemple de la figure 7, page 15. Dans cet exemple, les fautes ligne 1 et 2 se trouvent au premier niveau, leur profondeur est 0, la faute ligne 7 est au niveau 4 sa profondeur est 3.

Tableau des résultats d'activation
Le tableau 4, page 15 résume l'information issue de la phase d'activation concernant le statut de la faute celle-ci est codée 0 si la faute est non activée, 1 si la faute est activée et 2 si la faute est weak pour un test donné.

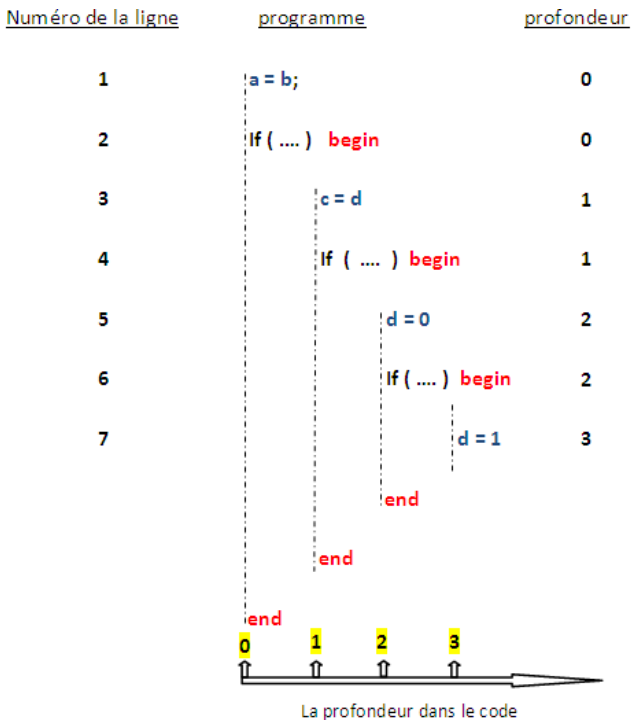

FIG. 7 – Exemple de la variable *profondeur*

	Test 1	Test 2	Test 3	. . .	Test j	. . .	Test p
Faute 1	1	1	1	. . .	1	. . .	0
Faute 2	1	0	2	. . .	0	. . .	0
Faute 3	1	2	0	. . .	1	. . .	1
.	.	.	.	. . .	.	. . .	.
.	.	.	.	. . .	.	. . .	.
.	.	.	.	. . .	.	. . .	.
Faute i	2	0	1	. . .	2	. . .	1
.	.	.	.	. . .	.	. . .	.
.	.	.	.	. . .	.	. . .	.
.	.	.	.	. . .	.	. . .	.
Faute n	0	2	0	. . .	0	. . .	1

TAB. 4 – Exemple d'une matrice des résultats d'activation

Deuxième partie
Exploration et description

La deuxième partie de ce rapport est l'occasion pour nous de présenter des méthodes d'analyse des données appliquées essentiellement aux tableaux de résultats d'activation du type illustré par le tableau 4, page 15. Au croisement de la ligne associée à chaque faute i et de la colone associée au test j figure un code qui représente le résultat de la phase d'activation. On peut considérer considère les fautes comme des individus et les tests comme variables qualitatives à trois modalitées (*non activée*, *activée* et *weak*) et traiter cette information comme un tableau croisé du type (Individus x Variables). Vu le grand nombre des fautes introduites dans les designs, nous avons donné plus d'importance aux tests.

5 Etude des similarités

Le grand nombre de fautes et de tests traités nous a ramené à étudier la similarité entre tests. Le but final de cette étude est d'arriver à distinger et exprimer les relations entre tests dans un design. Pour cela, avoir un indicateur de similarité puissant est très important.

5.1 Indicateurs de similarité

La construction des indicateurs de similarité retenus repose sur les informations disponibles après la phase d'activation. Une question fondamentale posée dans ce genre d'études est : comment peut-on définir une similarité entre deux tests ?

5.1.1 Indicateur de base

Le croisement de deux tests T_i et T_j $(i \neq j)$ est présenté dans le tableau ci-dessous (tableau 5, page 17). Où : Les cases A, E, et I représentent le nombre des caractéristiques communes.
De nombreux indices de similarité ont été proposés, notamment dans le cas des données binaires (absence ou présence), comme l'indice de *Jaccard* ou celui de *Russel et Rao*. Dans notre cas, avec l'aide des spécialistes du domaine, nous avons décidé de donner plus d'importance aux *fautes activées* et *weak* (cases E et I), en ignorant la case A qui représente les fautes *non activées*.

		T_i			
		non activées	activées	weak	total
T_j	non activées	A	B	C	N_{i0}
	activées	D	E	F	N_{i1}
	weak	G	H	I	N_{i2}
	total	N_{j0}	N_{j1}	N_{j2}	N

TAB. 5 – Tableau de croisement de deux tests

On obtient alors un indicateur de similarité *index*, donné par la formule suivante :

$$Index = \frac{E + I}{N - A}$$

5.1.2 Amélioration de l'indicateur *index*

Une information supplémentaire a été obtenue dans la phase d'activation. Il s'agit du premier temps d'activation. La disponibilité de cette information nous a donné l'idée de modifier et d'améliorer l'indicateur de base *index*, en proposant 3 modifications :

1^{ere} proposition

La première proposition n'est qu'une amélioration du l'indicateur *index*. Au lieu de s'intéresser aux fautes activées ou weak par les deux tests seulement, on rajoute le temps de la première activation comme un attribut de l'activation. Ainsi, dans le tableau ci-dessous nous avons codé 1 si les fautes sont activées en même temps 0 sinon.

		activées	
		0	1
activées	0	E_{00}	E_{01}
	1	E_{10}	E_{11}

TAB. 6 – Partition de la case E en quatre cases

Alors, la valeur E_{11} représente le nombre des fautes activées au même temps de première activation par les deux tests.
La valeur E_{00} représente le nombre des fautes activées par les deux tests mais à temps d'activation différents.

Le nouvel indicateur $IndexTime$ s'écrit donc sous la forme suivante :

$$IndexTime = \frac{E_{11} + I}{N - A}$$

Remarque : On vérifie simpelment l'inégalité :

$$IndexTime \leq Index$$

2^{eme} proposition

L'indice $IndexOrder$ est déduit uniquement du temps d'activation et consiste à donner un sens à l'ordre des fautes obtenues pour chaque test. Ainsi, deux tests sont parfaitement similaires si les fautes communément activées par ces deux tests sont ordonnées de même façon. Beaucoup d'algorithmes ont été appliqués pour calculer cet indice.
La faiblesse de $l'indexOrder$ est qu'il ne prend en compte que les fautes communément activées par les deux tests. On a une similarité parfaite entre les deux tests dans la figure ci-dessous, pourtant ils n'activent pas les mêmes fautes.

Test_1 :	**15**	**56**	50	51	**300**	**301**	**78**	34	100	3
Test_2 :	**15**	**56**	**300**	**301**	**78**	57	30			

FIG. 8 – Exemple de deux tests ordonnés par $firsttime$

3^{eme} proposition

On retient une autre façon d'utiliser l'ordre des fautes, mais cette fois on garde l'ensemble des fautes activées par les deux tests ordonnés par leur $firsttime$. Chacune représente une chaîne de caractères, $DistanceOrder$ calcule une distance entre deux chaines de caractères, existe plusieurs algorithme célèbres pour faire ce travail, parmi eux la distance de $Levenshtein$.
Cette dernière distance est très puissante par rapport à $l'indexOrder$.

5.1.3 Application sur le design $hc11verilog$

On rappelle que le design $hc11verilog$ contient 83 tests. Le calcul d'un indicateur de similarité donne les similarités entre 3403 couples de tests (soit

$p(p-1)/2$ avec $p = 83$). Pour comparer les différents indicateurs construits et déterminer celui ou ceux qui discriminent le mieux les tests, nous avons étudié minutieusement les distributions et les dispersions des quatre indicateurs de similarité. Un resumé de ces caracteristiques est fourni dans le tableau ci-dessous (tableau 7, page 19), et les histogrammes des indicateurs sont présentés en figure 9, page 19.

	Minimum	Maximum	Médiane	Moyenne	Ecart-type
Index	0.371	0.987	0.582	0.606	0.116
IndexTime	0.319	0.987	0.513	0.510	0.119
IndexOrder	0.784	1	0.975	0.946	0.586
DistanceOrder	4	341	224	210.82	68.61

TAB. 7 – Caractéristiques des différents indices de similarité ($hc11verilog$)

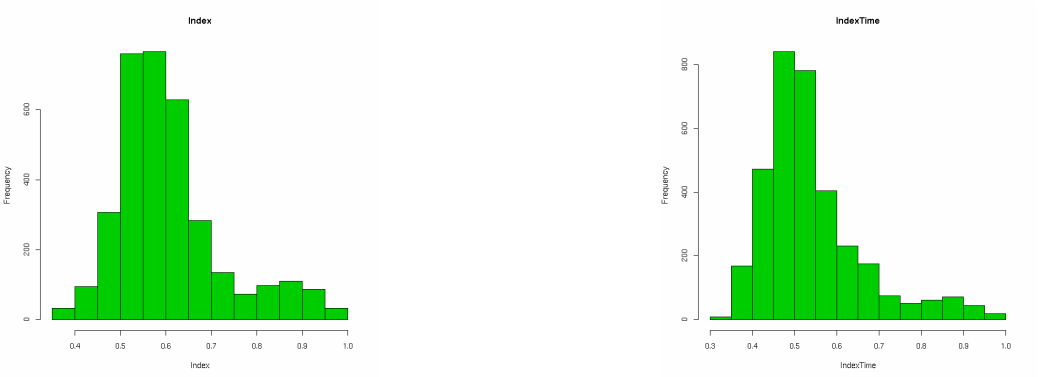

FIG. 9 – Histogramme *d'index* et *indexTime* sur le ($hc11verilog$)

On remarque une mauvaise distribution d'*indexOrder* avec un grand nombre de couples de tests avec similarité parfaite (figure 10, page 20). La discrimination entre tests en utilisant cet indicateur est très faible.

Firsttime joue un rôle très important dans la recherche des similarités entre tests, surtout pour les designs où les tests ont le même pouvoir d'activation des fautes.
Par exemple, dans le design *router2*, $74,18\%$ des fautes activées le sont par l'ensemble des tests. L'utilisation de l'indicateur *index* nous montre l'existance de deux groupes de tests et dans chaque groupe la similarité est très forte. Dans ce cas, *index* est moins discriminant, à l'inverse de *indexTime* qui donne une information plus détaillée sur les tests de chaque groupe, comme

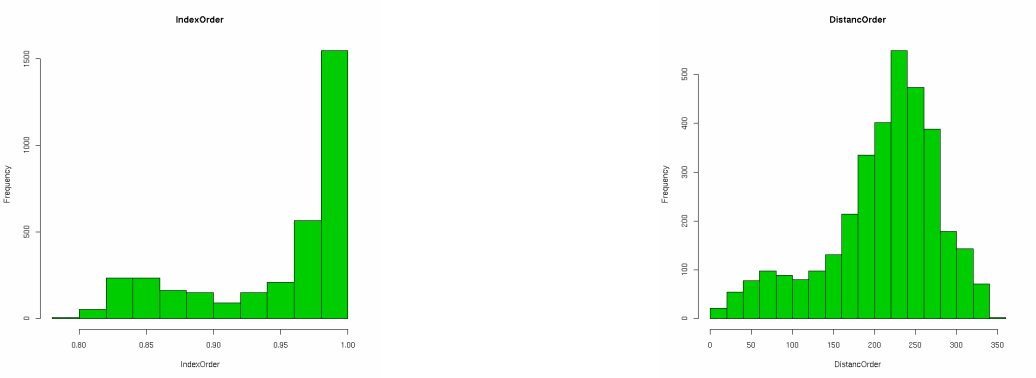

FIG. 10 – Histogramme *d'indexOrder* et *distancOrder* sur le *hc*11*verilog*

le montrent les histogrammes présentés en figure 11, page 20.

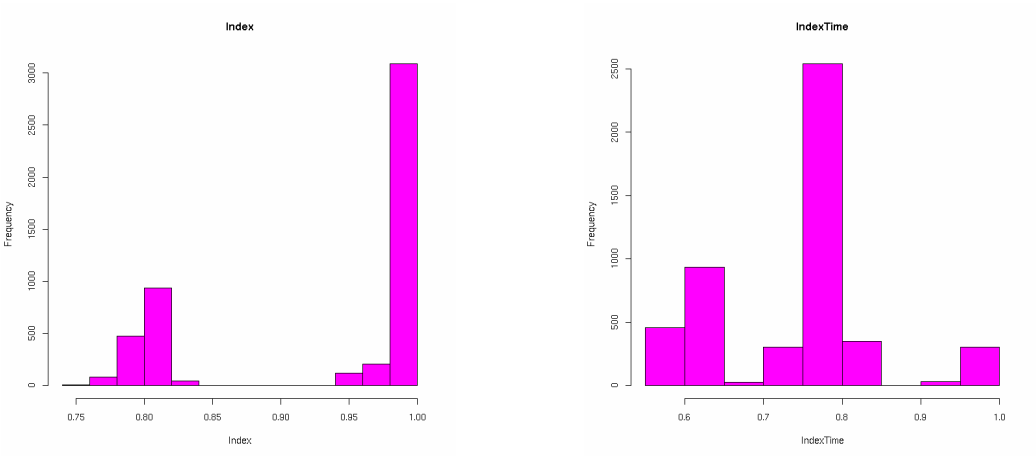

FIG. 11 – Histogramme *d'index* et *indexTime* pour le *router2*

5.1.4 Application de l'indicateur de similarité

Dans la pratique, l'indicateur de similarité a été utilisé dans l'algorithme de choix de tests dans la phase de détection. Si une faute n'est pas détectée par un test T_1 donné, il y a davantage de chances qu'elle soit détectée par un test T_2 très différent du premier, donc avec une faible similarité entre T_1 et T_2.

5.2 Classification et regroupement

Le calcul de $l'indexTime$ dans le cas du $hc11verilog$ a montré l'existence des fortes similarités entre quelques tests. Une classification hiérarchique des 83 tests a été réalisée en utilisant la méthode de $Ward$ sur le tableau des distances $(1 - indexTime)$. Le graphe dans la figure 12, page 21 représente le dendrogramme de la classification.

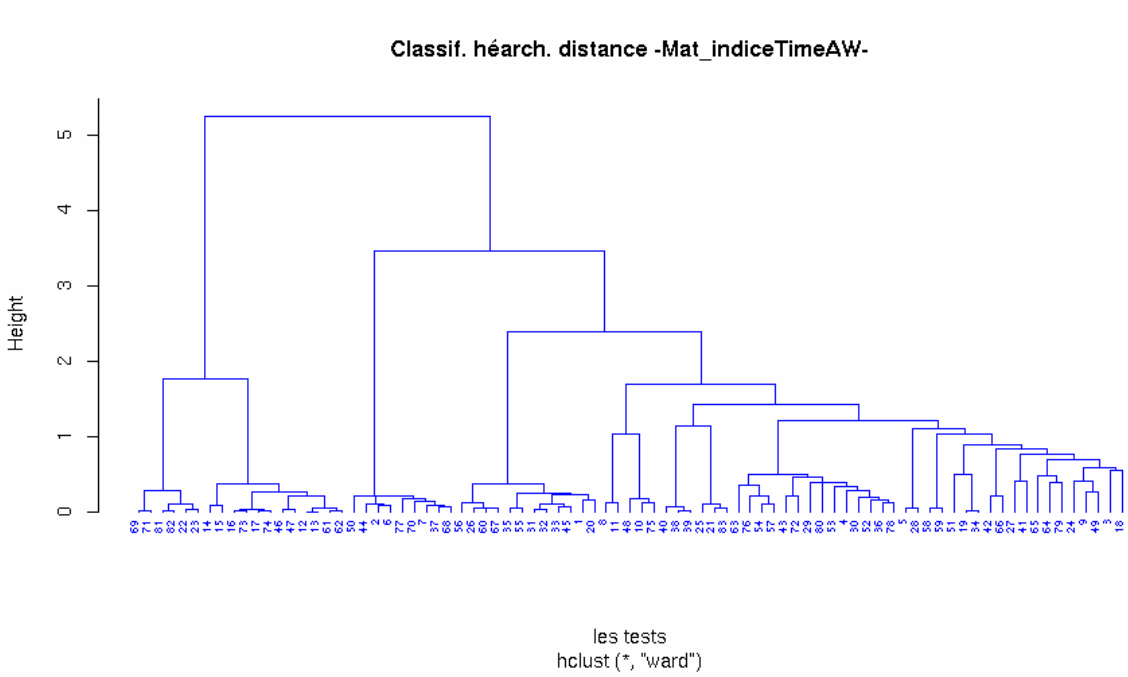

FIG. 12 – Arbre de Classification

La première remarque inspirée de cette classification est que les tests s'organisent dans deux grands groupes les (classes 1 et 2), l'un contient 18 tests et le deuxième contient 65 tests, et 8 groupes plus fins. Les groupes sont :

classe 1 :
classe1_1 : 69, 71, 81, 82, 22, 23
classe1_2 : 14, 15, 16, 73, 17, 74, 46, 47, 12, 13, 61, 62

classe 2 :
classe2_1 : 50, 44, 2, 6, 77, 70, 7, 37, 68
classe2_2 : 56, 26, 60, 67, 35, 55, 31, 32, 33, 45, 1, 20
classe2_3 : 8, 11, 48, 10, 75

classe2_4 : 38, 39, 40, 25, 21, 83
classe2_5 : 63, 76, 54, 57, 43, 72, 29, 80, 53, 4, 30, 52, 36, 78
classe2_6 : 5, 28, 58, 59, 51, 19, 34, 42, 66, 27, 41, 65, 64, 79, 24, 9, 49, 3, 18

Dans le même groupe, on remarque l'existence des tests voisins, par exemple le groupe classe1_2 qui contient les tests 12 à 17, puis les couples voisins (73,74), (46,47), (61,62), mais ce phénomène n'apparaît pas d'une manière régulière dans le $hc11verilog$.

De plus, dans la réalité, l'ordre des tests n'a aucun sens particulier. Le design *edac* nous montre bien ce phénomène des tests voisins, avec dix groupes de tests bien séparés et chaque groupe s'intéresse aux mêmes fautes. Cela s'explique par la nature du design lui même, qui contient des sous-designs indépendants, ce qui n'est pas le cas en général.

6 Analyse des correspondances multiples

L'objectif général de cette partie est de comprendre comment les tests et les fautes s'organisent dans un design en utilisant l'information disponible après la phase d'activation. Nous avons appliqué une méthodes d'analyse descriptive multidimentionnelle : *l'analyse des correspondances multiples* (ACM). Cette méthode est particulièrement bien adaptée à l'exploration d'enquêtes ou au traitement des variables catégorielles, ce qui le cas ici.

6.1 Principe de la Modélisation ACM

constitution du tableau de données

L'ACM est une technique de description euclidienne de données qualitatives : on considère ici n fautes décrites par p tests T_1,T_2,T_3, ... ,T_p. Chaque test T_j $(j = 1, \ldots, p)$ a $m_j = 3$ modalités de réponse, définissant ainsi 3 catégories : 0 pour les fautes non acvtivées, 1 pour les activées et 2 pour les weak.

Chaque faute est décrite par les numéros des catégories des p tests auxquelles elle est soumise. Ces données brutes se présentent donc sous forme d'un tableau à n lignes (les fautes) et p colonnes (les tests). En juxtaposant les p tableaux d'indicatrices de chaque test, on obtient un tableau X de $3p$ colonnes, comme le montre le tableau ci-dessous (tableau 8, page 23).

	Test 1			Test 2			.
	non activée	activée	weak	non activée	activée	weak	. . .
Faute 1	0	1	0	1	0	0	. . .
Faute 2	1	0	0	0	1	0	. . .
Faute 3	0	0	1	0	1	0	. . .
.	.	.	.	.	.	.	. . .
.	.	.	.	.	.	.	. . .

TAB. 8 – Table disjonctif complet des tests

Ce tableau est dit disjonctif complet car pour chaque test T_j la faute i est repérée par une chaîne de 3 chiffres $x_i^j \in \{0,1\}$ comprenant au plus un (1) (caractère disjonctif) et au moins un (1) (caractère complet).

L'ACM est l'ACB du tableau de BURT

On note B le tableau croisé B=X'X obtenu par produit matriciel et qui a pour particularité d'être décomposé en bloc B_{ij} ; $i, j = 1, \ldots, p$. Chaque bloc B_{ij} est en fait le tableau de contingence de taille 3×3 croisant les résultats des tests T_i et T_j . Ce tableau, dit tableau de BURT, est un tableau symétrique et résume les liaisons deux à deux existant entre tests. Le tableau 9, page 23 illustre la situation :

		T_1			T_1			.
		0	1	2	0	1	2	. . .
	0	1931	0	0	1791	132	8	. . .
T_1	1	0	110	0	16	82	12	. . .
	2	0	0	99	20	35	44	. . .
	0	1791	16	20	1827	0	0	. . .
T_2	1	132	82	35	0	249	0	. . .
	2	8	12	44	0	0	64	. . .
	.	.	.	.	.	.	.	. .
.	.	.	.	.	.	.	.	. .
	.	.	.	.	.	.	.	. .

TAB. 9 – Extrait du Table de BURT du *l'hc*11*verilog*

Soit D le tableau diagonal des effectifs marginaux des $3p$ catégoris. C'est le tableau bloc diagonal formé à partir des tris à plats de chaque test puisque $D_j = B_j j; j = 1, \ldots, p$

$$\mathcal{D} = \begin{pmatrix} D_1 & 0 & \cdots & 0 \\ 0 & D_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_p \end{pmatrix}$$

Le tableau de profils lignes associées à B est donc $(pD)^{-1}B$, et le tableau des profils colonnes est $B(pD)^{-1}$. L'analyse des correspondances (ACB) de B revient à diagonaliser :

$$(\frac{1}{p}D^{-1}B)^2$$

On montre classiquement. (cf. Référence bibliographique numéro 1) que la recherche des éléments propres de B par cette ACB conduit aux mêmes vecteurs propres que $\frac{1}{p}D^{-1}B$, mais associés à des valeurs propres égales à λ^2. λ^2 s'interprète comme un indicateur de la part de liaison entre tests et fautes qui est prise en compte par le vecteur propre associé

6.2 Application au design *hc11verilog*

Une première remarque avant l'application de l'ACM est qu'il existe des fautes à une modalité unique c'est-à-dire des fautes non activées ou activées ou weak par tous les tests. La bonne réalisation de l'ACM exige l'élimination de ce genre de fautes qui ne donne aucune information differenciant les tests à part qu'elles sont d'un statut unique.

Le *hc11verilog* contient 150 fautes non activées, 13 fautes weak et 233 fautes activées par tous les tests. Ces fautes ne participent pas à la construction de l'ACM. Donc le nombre des tests restant après suppression de ces fautes est $p = 83$ et le nombre des fautes est $n = 2140$.

Le grand nombre des fautes non activées pour tous les tests se voit clairement dans le graphique ci-dessous (voir figure 13, page 25), il est au minimum 82.4% pour le cinquième test et 94.1% au maximum pour le test 78, et cette inéquivalence du poids aura un effet sur la réalisation de l'ACM.

6.2.1 Inerties et coordonnées

Le nombre des modalités actives est 249 qui conduit à 166 facteurs produit par l'ACM ($\sum_{i=1}^{p} m_i - p$) et à une inertite (variance) totale de $\frac{249}{83} - 1 = 2$. La moyenne des 166 valeurs propres vaut 0.012, $(1/p)$. Cette quantité joue le rôle d'un seuil d'élimination des axes correspondant à des valeurs propres

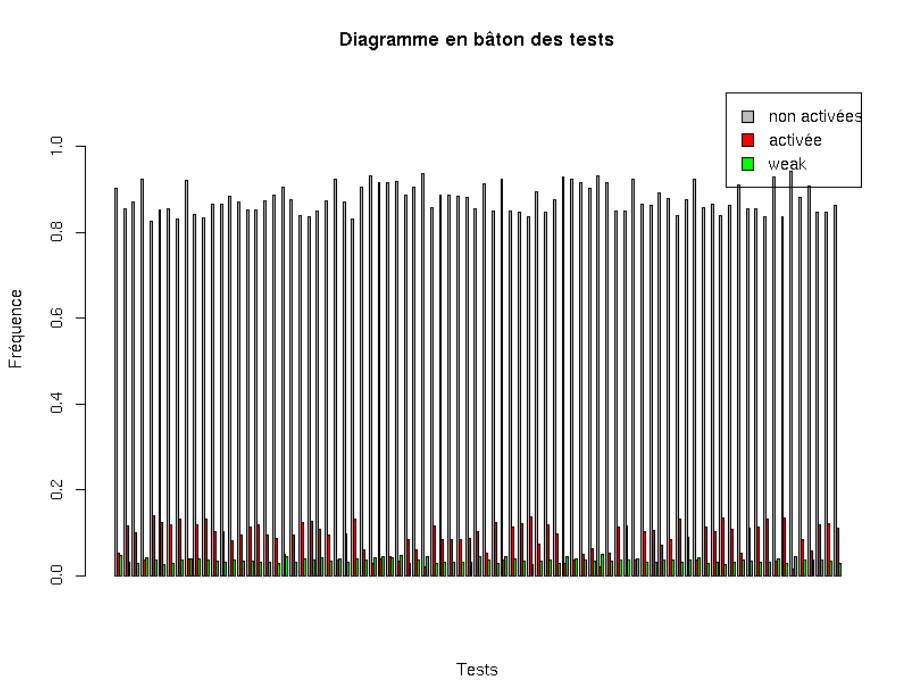

FIG. 13 – Diagramme en bâton des tests

inférieures. On calcule pour le reste des valeurs propres la quantité $\lambda_{corrige}$ donnée par la formule suivante :

$$\lambda_{corrige} = \left[\frac{p}{p-1}(\lambda - \frac{1}{p})\right]^2$$

afin de déterminer le nombre de dimensions informatives à retenir. Les 4 premières valeurs propres et sont présentées dans le tableau 10, page 25 :

	λ_i	Pourcentage	Pourc. cumulé		λ_i corrigé	Pourcentage	Pourc. cumulé
1	0.738	36.92	36.92		0.541	79.37	79.37
2	0.356	17.81	54.73		0.121	17.81	97.18
3	0.095	4.75	59.48		0.007	1.04	98.22
4	0.085	4.28	63.76		0.0056	0.82	99.03

TAB. 10 – Table d'inerties expliquées par les quatre premiers axes

Le critère $\lambda > \frac{1}{p}$ conduit à ne retenir que deux axes, avec une inertie expliquée corrigée très forte, puisque égale à 97.18%. Le diagramme des valeurs propres (figure 14, page 26) montre lui aussi une chute après λ_2. On a donc décidé d'interpréter ici uniquement les deux premiers axes. On néglige ainsi moins de 3% de l'information portée par les données.

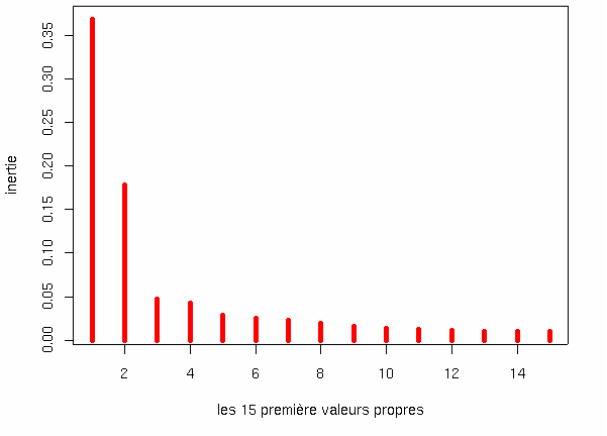

FIG. 14 – diagramme des valeurs propres

Les vecteurs propres correspondant aux deux premières valeurs propres (non triviales) représentent les coordonnés des modalités dans le plan factoriel, et on peut obtenir les coordonnées des fautes dans le même plan, par la relation *barycentrique*.

6.2.2 Interprétation des résultats

Une première remarque s'impose : le plan sépare bien les trois catégories de réponse d'un test, donc on n'a pas de mélange entre modalités. Cela s'explique par le même comportement général des tests vis-à-vis des fautes pour le $hc11verilog$. Le regroupement des modalités *non activée* traduit le fait qu'un grand nombre des fautes sont non activées pour tous les tests.

Le premier axe oppose les modalités *non activée* aux *activée* et *weak*. Les fautes *weak* contribuent fortement à l'axe 1 alors que les fautes activées contribuent plutôt à l'inertie de l'axe 2. De plus, il n'y a pas une grande différence de contribution à l'intérieur de chaque groupe de modalités.

Le premier axe ordonne les fautes suivant le nombre de tests où la faute est *weak*. Le deuxième donne plus d'importance au nombre de tests qui activent la faute. Les fautes dans le plan factoriel (figure 16, page 28) construisent un triangle imparfait surtout entre l'angle des *weak* et les *non activée*. Cela nous montre que les fautes *weak* par un grand nombre de tests ont une tendance à être *activées* par le reste des tests plutôt qu'à être *non activées*.
Les fautes dans ce design sont moins partagées entre les trois modalitées, ce

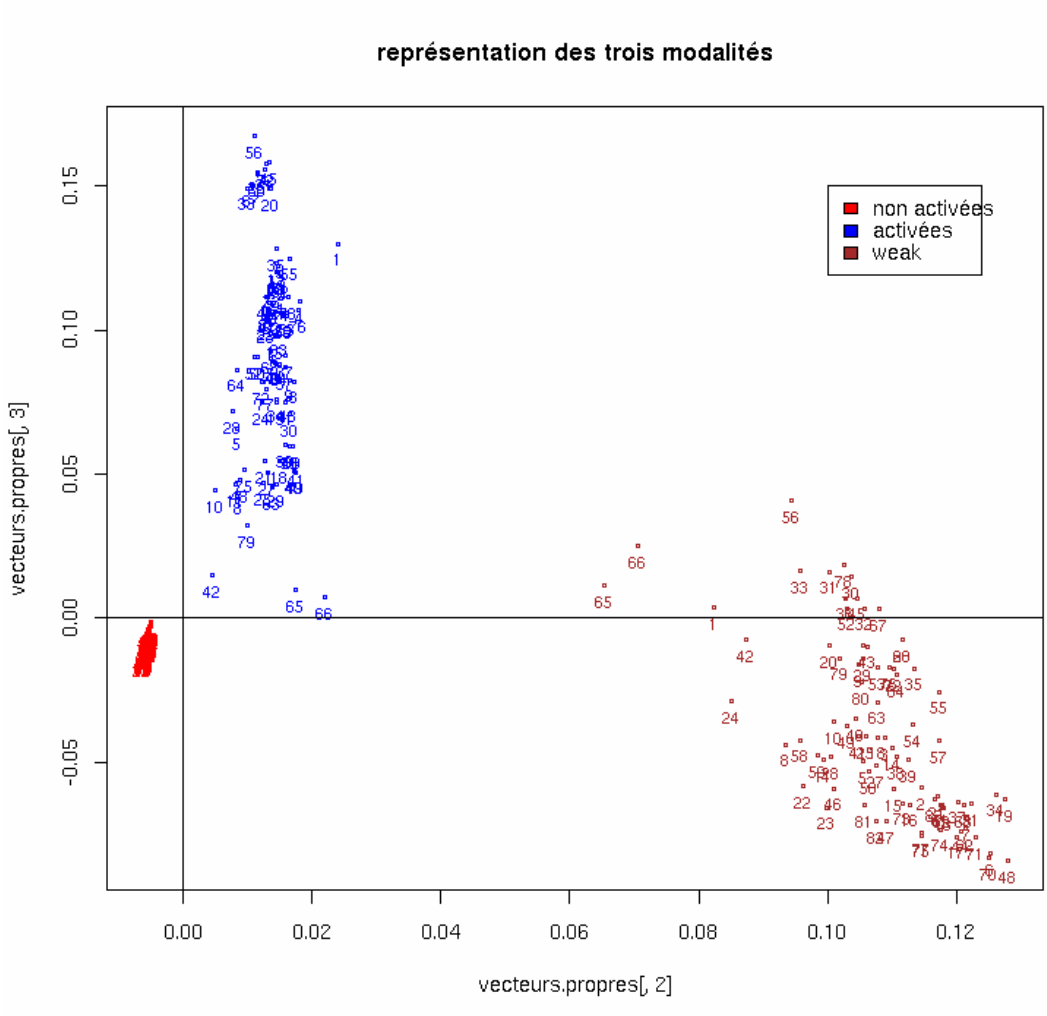

FIG. 15 – représentation des tests sur les deux premiers axes

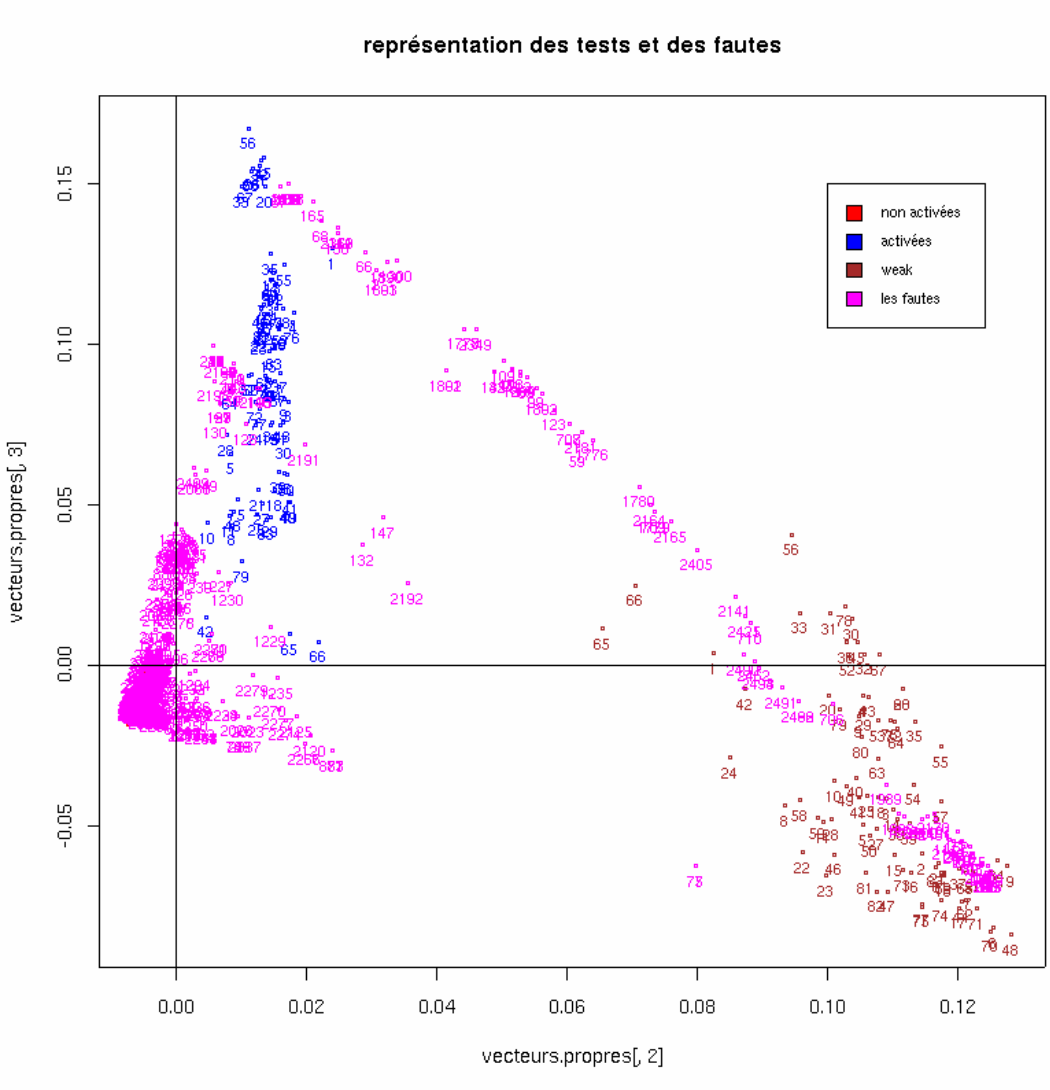

FIG. 16 – représentation des fautes sur les deux premiers axes (*hc*11*verilog*)

qui se traduit par l'absence des fautes à l'intérieur de triangle.

Pour mieux comprendre les positions des tests dans le plan factoriel nous avons regardé plus en détail quelques tests :

Le test 56 active les fautes activées par un grand nombre de tests (figure 17, page 29). Ce test n'a pas de spécificité concernant l'activation.

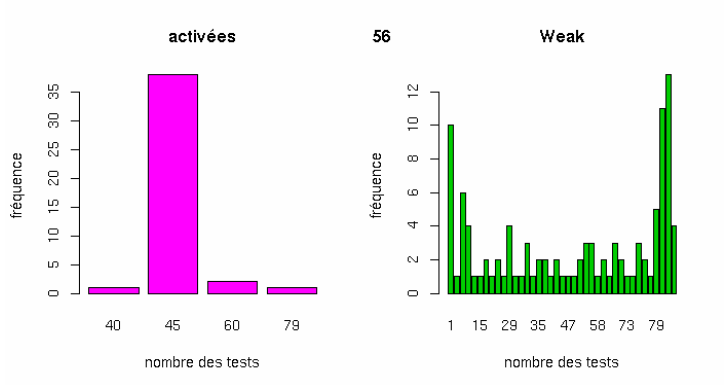

FIG. 17 – Diagramme en bâton des fautes activées et weak pour le test 56

La plupart des fautes activées pour le test 65 sont activées par peu de tests. Le comportement est similaire pour les weak (figure 18, page 29).

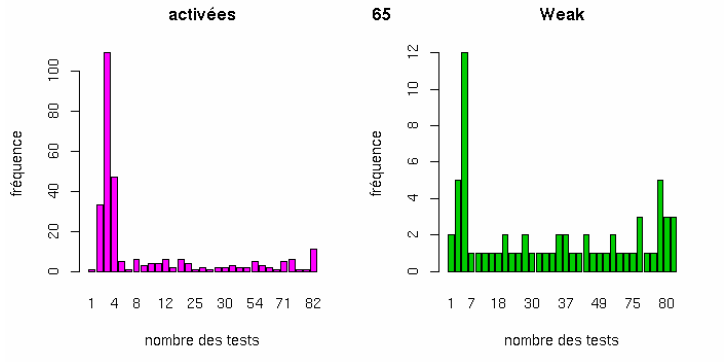

FIG. 18 – Diagramme en bâton des fautes activées et weak pour le test 65

Les faute weak pour le test 48 sont weak pour beaucoup de tests (figure 19, page 30).

L'identification des groupes de tests

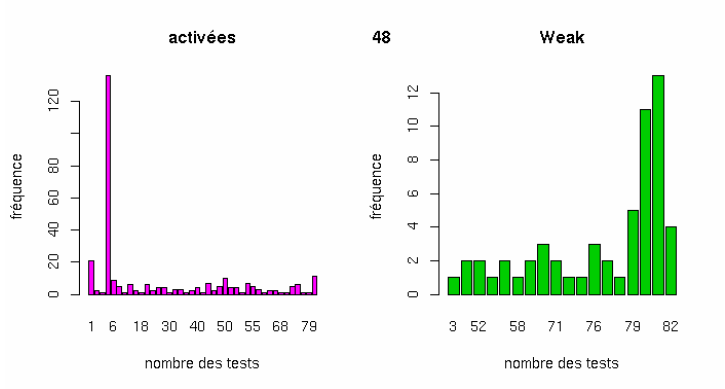

FIG. 19 – Diagramme en bâton des fautes activées et weak pour le test 48

Par un effet de couleurs, nous avons fait apparaître les groupes de tests construits après l'étude de similarité par classification automatique (figure 20, page 31). Les groupes classe2_2, classe2_5 et classe2_6 paraissent plus dispersés que les autres.

6.3 Traitement des variables supplémentaires

Dans cette partie, on étudie certaines variables supplémentaires (types de fautes, firsttime), toujours sur le composant *hc*11*verilog*.

Les variables actives sont celles qui déterminent les axes. Les variables supplémentaires ne participent pas au calcul des valeurs propres et vecteurs propres mais peuvent être représentées sur les plans factoriels selon le principe *barycentrique* dans le cas où ce sont des variables qualitatives : chaque catégorie est le point-moyen d'un groupe d'individus qui a choisi cette catégorie comme modalité de réponse.

6.3.1 Types des fautes

La variables *type-faute* est une variable nominale de dix catégories (pour le *hc*11*verilog*). Le graphique (figure 21, page 32) nous montre comment se présente *type-faute* dans ce design.

La représentation de cette variable dans le plan factoriel est montrée sur la figure 22, page 33. Les fautes *control-flow* (en rouge) et les *autres-fautes* (en bleu) ne sont pas mélangées, mais elle ne sont pas parfaitement séparées dans ce plan où la modalité *condition-false* se trouve en opposition avec le

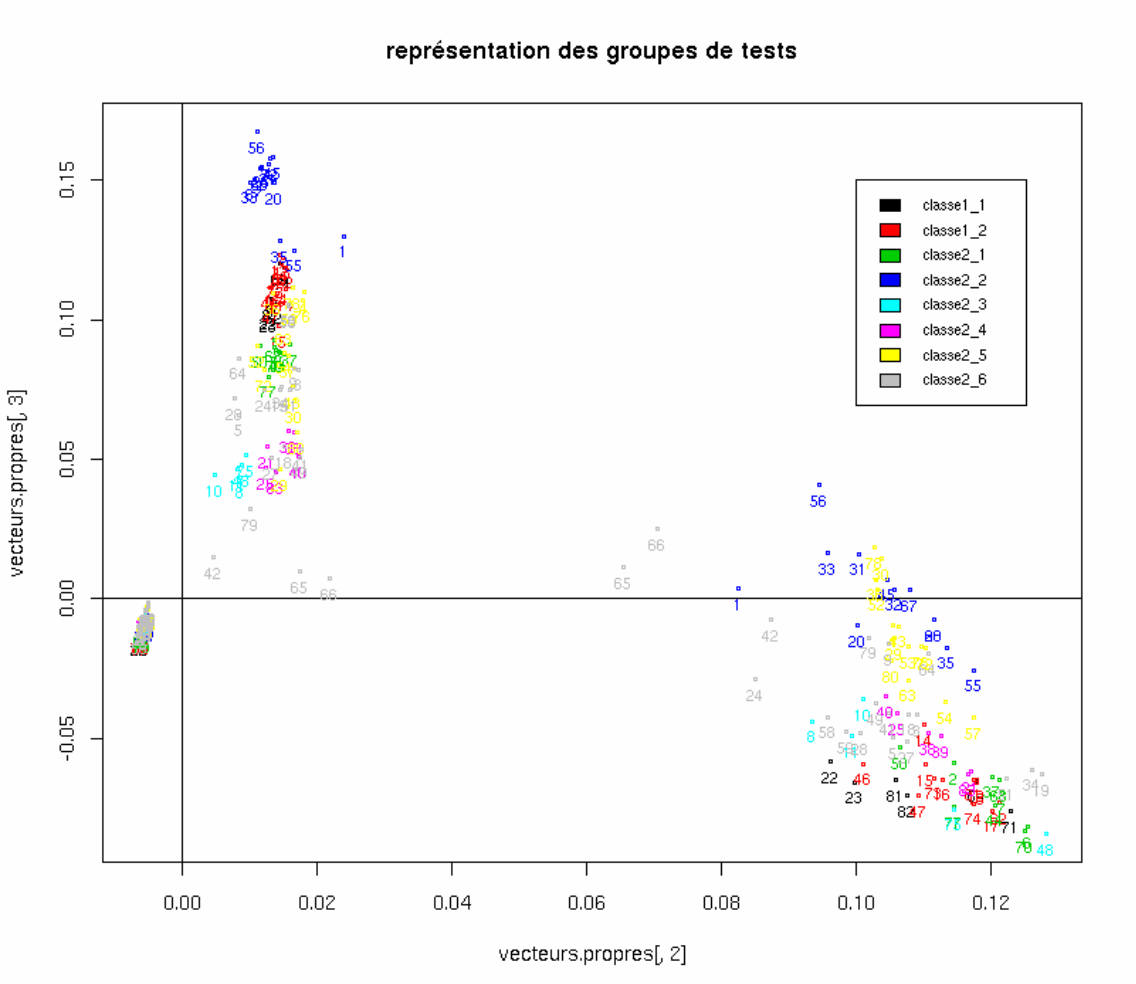

FIG. 20 – Représentation des groupes de tests dans le plan factoriel

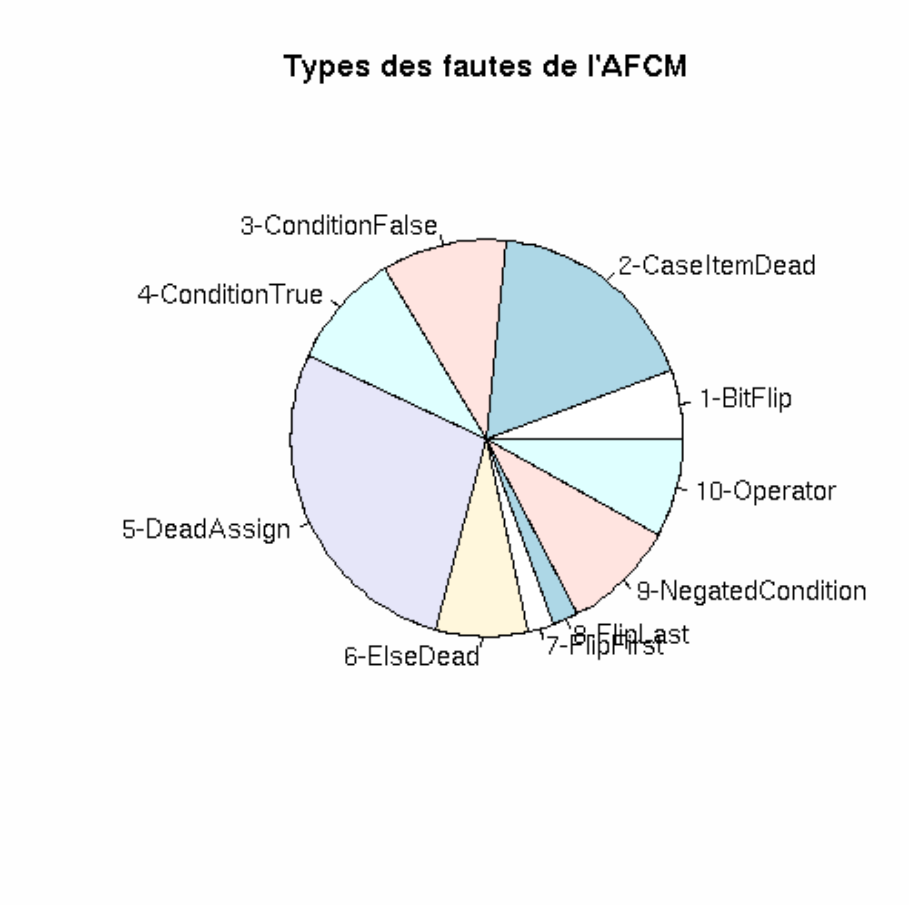

FIG. 21 – La présence des types des fautes dans le *hc*11*verilog*

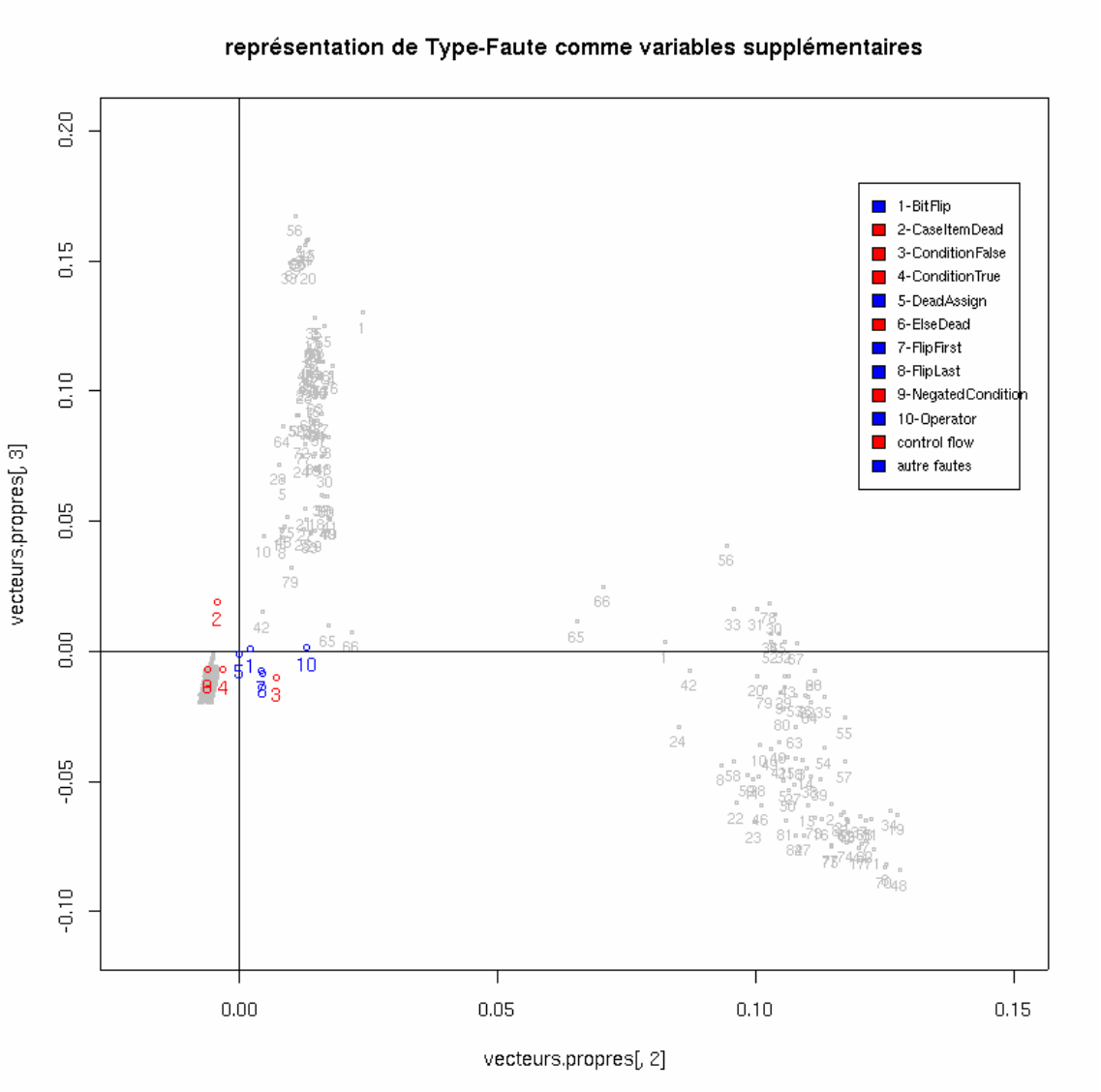

FIG. 22 – Présentation des type des fautes dans le plan factoriel

reste de groupe *control-flow* par rapport à l'axe 1, la modalité *CaseItemDead* oppose le reste de sont groupe par rapport à l'axe 2.

On remarque aussi que *ElseDead* et *NegatedCondition* sont les plus proches des *non activées*, cela s'explique par le fait que les fautes de ce type soient non activées par plus de tests plutôt qu'activée ou weak.

6.3.2 Firsttime

On rappelle que chaque faute activée par un test a un temps d'activation correspondant. L'utilisation de $firsttime$ comme variable supplémentaire est très compliquée parce qu'on est devant 83 variables, avec un mélange de données qualitatives (pour les fautes *non activées* ou *weak*) et quantitatives pour les *activées*.

Malgré cette difficulté, nous avons présenté ces variables en gardant les deux modalités (*non activées* ou *weak*), et construit des classes de temps pour le reste. Ainsi recodée, la variable $firsttime$ devient utilisable en tant que variable qualitative.

Les classes sont construites de la même façon pour tous les tests afin de garder une certaine homogénéité de mesure. On a adopté une méthode pour minimiser la perte d'information en passant de la variable quantitative à la variable ordinale. Les variables finales recodées contiennent entre 3 et 7 modalités dont les deux initiales (*non activée* et *weak*). Les classes sont :

$$\begin{aligned}
&\text{classe } 1 : \ firsttime \ \leq 100 \\
&\text{classe } 2 : 100 < \ firsttime \ \leq 1100 \\
&\text{classe } 3 : 1100 < \ firsttime \ \leq 2500 \\
&\text{classe } 4 : 2500 < \ firsttime \ \leq 3900 \\
&\text{classe } 5 : \ firsttime \ > 3900
\end{aligned}$$

Remarque : Cette décomposition n'est valable que pour le composant $hc11verilog$ et devra être adaptée au cas par cas pour d'autres designs.

La représentation des 83 variables $firsttime$ correspondant aux 83 tests dans le plan factoriel (figure 23, page 35), nous montre la dispersion des tests à ce niveau d'information et nous a permis de voir que cette variable a un effet d'ordre à l'intérieur de chaque groupe de test.

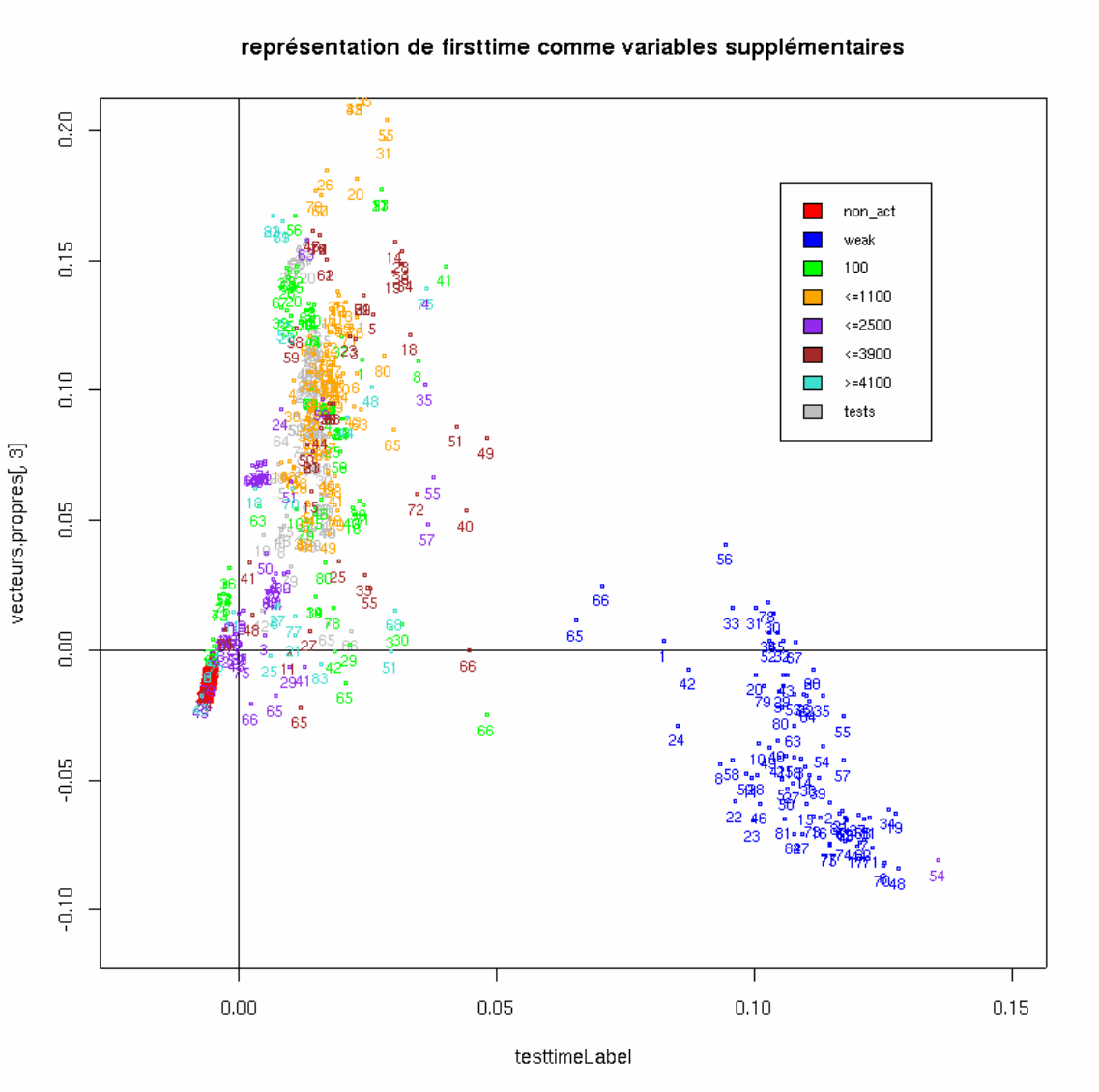

FIG. 23 – Présentation de $firsttime$ dans le plan factoriel

Pour cela, on a préféré associer une représentation séparée à chaque groupe de tests, pour permettre une vision microscopique. Par exemple, le groupe $classe1_2$ qui contient les tests 12, 13, 14, 15, 16, 17, 73, 74, 46, 47, 61 et 62 est présenté dans la figure 24, page 36. Dans ce graphique, on voit que les tests sont bien ordonnés par $firsttime$, et le groupe garde son homogénéité pour presque tous les tests à part le $14^{ème}$ et le $15^{ème}$ test dans la $4^{ème}$ classe.

Mais en regardant un autre groupe de tests par exemple le groupe $classe2_1$ (figure 25, page 37), on trouve un autre ordre des tests qui distingue ce dernier par rapport a la $classe1_2$.

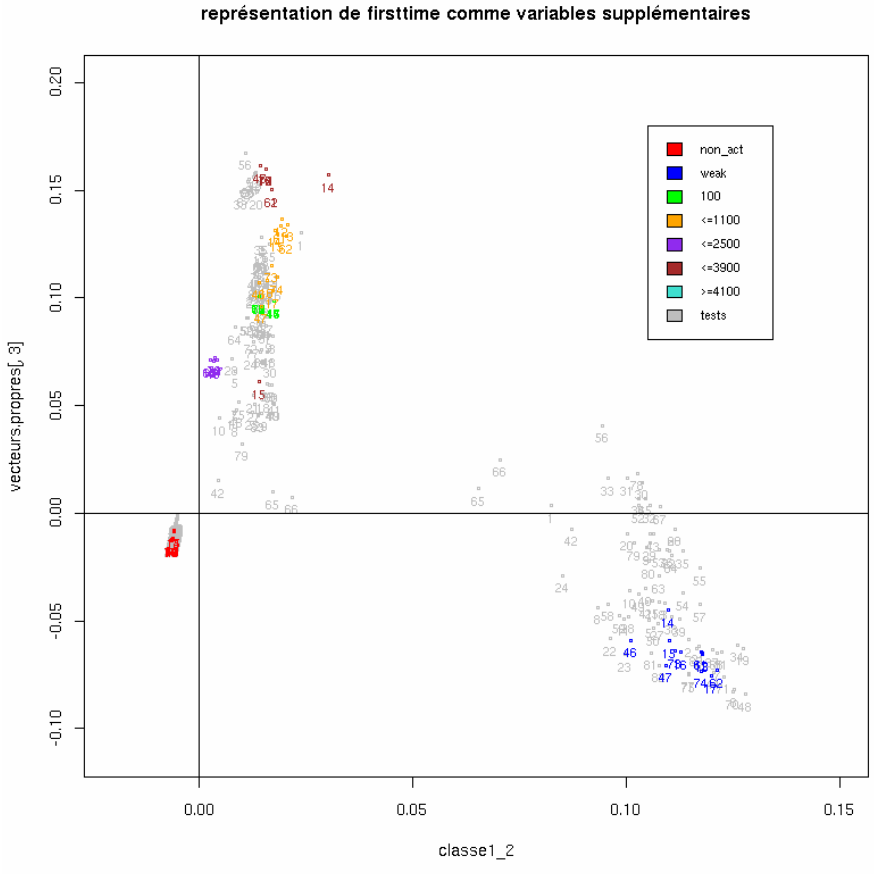

FIG. 24 – Présentation de $firsttime$ de $classe1_2$ dans le plan factoriel

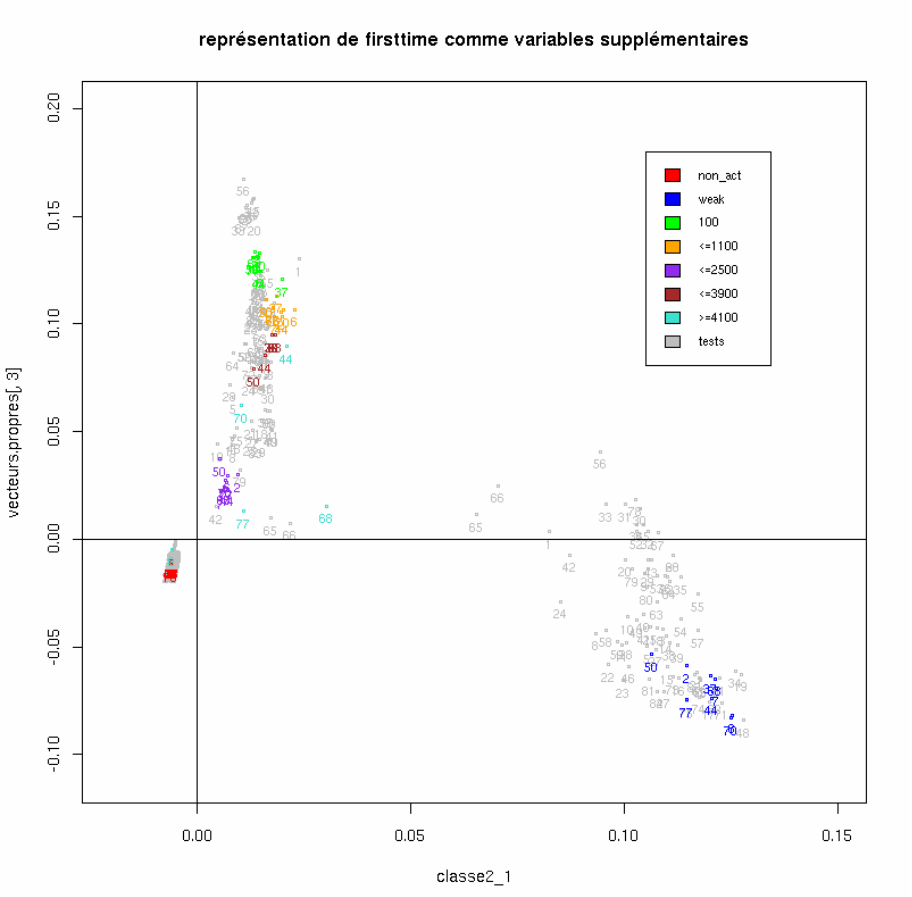

FIG. 25 – Présentation de $firsttime$ de $classe2_1$ dans le plan factoriel

6.4 Les autres designs

L'application de l'ACM au cas des composants *edac* et le *router2* n'apporte pas grand chose. De fait tout d'abord de la structure très particulière du premier et d'autre part le grand nombre de fautes activées par tous les tests dans le deuxième design. Par contre sur le design $i8051$ nous avons obtenu les résultats qui semblent plus riches. En particulier nous avons observé :

– Un taux d'activation très élevé pour certains tests, surtout le test numéro 4 (active 93.59% parmi les 2592 fautes gardées pour l'analyse) (voir la figure 26, page 38).

– Dans le premier plan factoriel, les modalités du quatrième test sont isolées par rapport aux autres catégories (figure 27, page 39).

– Les fautes dans ce design couvrent la totalité de la surface du triangle. Cela est dû au nombre restreint des tests et à leur pouvoir d'activation dans ce design, à l'inverse du *hc11verilog*.

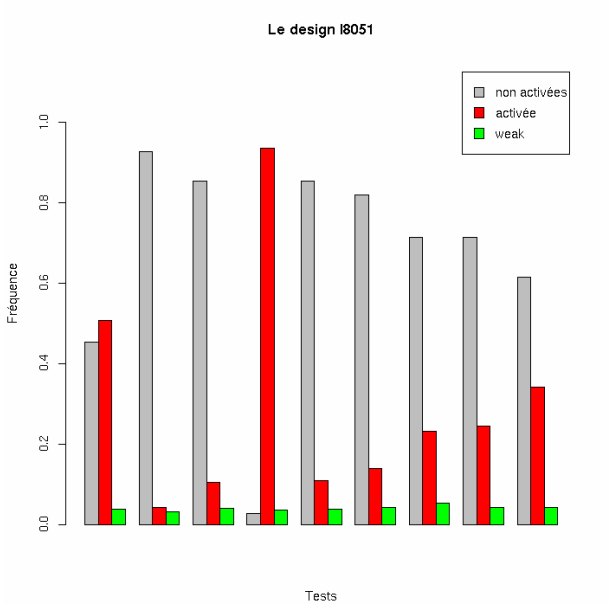

FIG. 26 – Diagremmes en bâtons des tests (i8051)

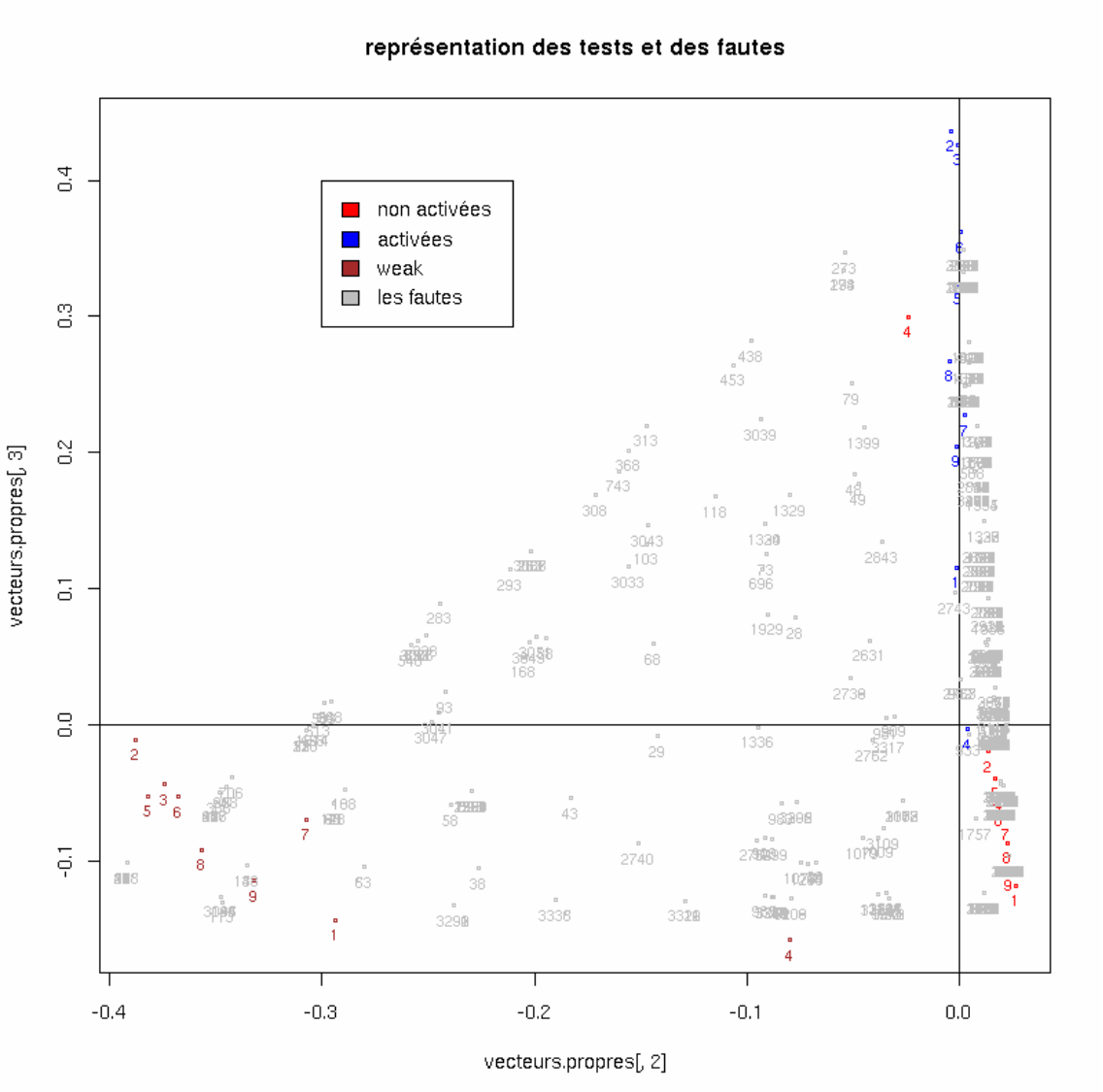

FIG. 27 – Les fautes et les tests dans le plan factoriel pour le $i8051$

Troisième partie

Modélisation et Prévision de la métrique

Dans la troisième partie du rapport, nous avons essayé de modéliser la méthodologie de détection et de proposer une méthode probabiliste qui permettre d'estimer la métrique.

Soit A_i l'ensemble des tests qui activent la faute F_i, pour tout $i = 1, \ldots, N$ où N est le nombre de fautes dans un design.

$$A_i = \{T_j \setminus T_j \text{ active } F_i, \forall\, j = 1, \ldots, p\}$$

Et soit D_i l'ensemble des tests qui détectent la faute F_i, parmi ceux qui l'activent.

$$D_i = \{T_j \setminus T_j \text{ détecte } F_i, \forall\, T_j \in A_i\}$$

Clairement, $D_i \subseteq A_i$ ($\forall$i=1,...,N)

Dans cette partie, nous nous sommes intéressés uniquement aux fautes activées par au moins un test. Nous considérons une faute F_i activée dans le design si et seulement si l'ensemble A_i n'est pas vide. L'ensemble fini de ces fautes constitue la population des fautes activées, notée Ω dans un design.

$$\Omega = \{F_i \setminus A_i \neq \emptyset, \forall\, i = 1, \ldots, N\}$$

La phase de *détection* se déroule sur cet ensemble Ω.

Soit Ω_{det} l'ensemble des fautes détectées par au moins un test, parmi ceux qui l'activent.

$$\Omega_{det} = \{F_i \setminus D_i \neq \emptyset, \forall\, F_i \in \Omega\}$$

La *détection* est une relation ($Test$, $Faute$) entre l'ensembles des fautes F_i et l'ensemble des tests $T_j \in A_i$. Si $T_j \in D_i$ la faute a un statut final *détectée*, sinon le statut de la faute est *non détectée*, et ce n'est pas un statut final. Le statut *non détectée* de la faute F_i devient final si tous les tests de A_i sont passés.
On défini alors la métrique d'une batterie de tests par le rapport :

$$\textit{Métrique} = \frac{\text{card}(\Omega_{det})}{\text{card}(\Omega)}$$

Estimer la probabilité qu'une faute soit détectée $F_i \in \Omega_{det}$ est une autre façon d'estimer *la métrique*.

7 Analyse Logistique

Calculer la métrique à partir d'un échantillon de fautes tiré au hasard nous donne un mesure globale de la capacité de détection d'une batterie de tests. Avec le même principe d'échantillonnage et avec un modèle de la régression logistique on peut obtenir en plus la probabilité qu'une faute soit détectée conditionnellement à certaines connaissances préalables disponibles sur la faute.

A l'aide de la connaissance des variables explicatives $X^1, \ldots, X^q$, la régression logistique fournit la probabilité conditionnelle à ces variables qu'a une faute d'être détectée : $P\left[F_i \in \Omega_{det} \mid X_i\right]$. La métrique s'obtient empiriquement par la formule :

$$\text{Métrique} = \frac{1}{N} \sum P\left[F_i \in \Omega_{det} \mid X_i\right]$$

Et l'on espère estimer la métrique en optimisant la taille de l'échantillon, et pourquoi pas, corriger la proportion des fautes détectées dans l'échantillon.

7.1 Principe de la régression logistique

7.1.1 Définitions

La régression logistique se définit comme une technique permettant d'ajuster une surface de régression à des données lorsque la variable est dichotomique. Cette technique n'exige pas une distribution normale des prédicteurs ni l'homogénéité des variables. De plus, les variables indépendantes (prédicteurs) peuvent être des variables dichotomiques ou continues.

L'équation de la régression logistique est donnée par :

$$Ln(\frac{\widehat{\pi}}{1-\widehat{\pi}}) = \alpha + \sum \beta_j X_{ij}$$

Où $\widehat{\pi}$ est la probabilité d'apparition a un groupe.
Dons cette équation, $Ln(.)$ correspond au log népérien et la côte $\frac{\widehat{\pi}}{1-\widehat{\pi}}$ est le rapport de la probabilité de faire partie d'un groupe par la probabilité de

ne pas faire partie du groupe. En fait, cette équation fait partie d'une famille de modèles qu'on appelle *modèles linéaires généralisés* spécifié par la donnée d'une fonction de lien particulière : le $Logit = Ln(\widehat{\pi}/(1-\widehat{\pi}))$. Ce modèle correspond à une variable réponse dont la loi de probabilité est la loi *Binomiale* et le modèle est caracterisé par une fonction de prédiction linéaire $\eta(x) = \alpha + \sum \beta_j X_{ij}$.

L'estimation des paramètres du modèle de régression logistique se fait généralement par la méthode du maximum de vraisemblance. Pour cela, on écrit la vraisemblance de l'échantillon. Lorsque les observations individuelles y_i (i=1,...,n) sont supposées indépendantes, cette vraisemblance s'écrit comme :

$$L(\alpha, \beta_j) = \prod_{i=1}^{n} [P(y=1|x, \alpha, \beta_j)]^{y_i} [1 - P(y=1|x, \alpha, \beta_j)]^{1-y_i}$$

Soit Y la variable de réponse (à prédire) :

$$y = \begin{cases} 1 & \text{si la faute est détectée, avec probabilité } \pi \\ 0 & \text{si la faute n'est pas détecté avec, probabilité } 1-\pi \end{cases}$$

7.1.2 Stratégie de la modélisation

La régression logistique peut s'effectuer de diverses façons, selon l'objectif attendu de cette étude (descriptif, explicatif ou prédictif). Dans notre cas, la réalisation de la régression logistique est pour un objectif prédictif. Pour cela, une stratégie qui a été adoptée consiste à privilégier la qualité des estimateurs et des prédicteurs. Ceci conduit à rechercher des modèles *parcimonieux* c'est-à-dire avec un nombre restreint de variables explicatives.

Un bon modèle n'est donc plus celui qui explique le mieux les données au sens d'une déviance minimale au prix d'un nombre important de variables pouvant introduire des colinéarités. Le bon modèle est celui qui conduit aux prédictions les plus fiables.

Le critère d'information d'Akaïke (**AIC**) est l'un des critères utilisés pour mesurer la qualité du modèle. Il s'applique à tout modèle estimé par maximisation d'une log-vraisemblace L. Dans notre cas le choix de modèle fut fondé sur ce critère.

$$AIC = -2L + 2\frac{d}{n}$$

Où d est le nombre des paramètres du modèle (nombre de variables plus un) et n est le nombre d'observations.

7.2 Analyse et traitement des prédicteurs potentiels

Avant de se lancer dans la modélisation logistique, construire des prédicteurs potentiels a été la première étape de l'étude. Après beaucoup de réflexion et de travail avec les spécialistes du domaine, les variables explicatives candidates à l'analyse sont représentées dans le tableau ci-dessous (tableau 11, page 43) :

	Variable	Type de la variable
X_1	*Type-faute*	qualitative
X_2	*Nb-test-act*	quantitative
X_3	*Nb-test-weak*	quantitative
X_4	$Profondeur$	quantitative
X_5	*Poids-faute*	quantitative

TAB. 11 – Table des prédicteurs potentiels

Nb-test-act : est le nombre de tests qui activent la faute.
Nb-test-weak : représente le nombre du tests où la faute est weak.
Les autres variables ont déjà été présentées dans la première partie.

7.2.1 Fiabilité et pertinence des variables

Une étude détaillée des co-variables a été faite sur les quatre designs disponibles et particulièrement le composant *hc11verilog* et *i8051*, pour vérifier leur fiabilité statistique et également leur pertinence par la modélisation logistique. L'analyse a été faite sur toutes les fautes activées (Ω) pour répondre sur la question suivante :
quels sont les critères pertinents qui discriminent entre les fautes détectées et non détectées ?

Hc11verilog

Les variables quantitatives
Le tableau ci-dessous représente les caractéristiques de dispersion et de tendance centrale des variables quantitatives initiales :

Variable	min.	max.	mediane	moyenne	écart-type
Nb-test-act	1	83	3	15.16	26.12
Nb-test-weak	0	82	0	2.59	12.32
Profondeur	0	21	6	7.97	5.42
Poids-faute	1	1027	6	81.68	229.05

TAB. 12 – Caractéristiques des variables quantitatives

Les histogrammes et les box-plots (figure 28, page 44) nous donnent une idée de la distribution des variables. On peut en particulier constater la mauvaise distribution des variables (*Nb-test-act*, *Nb-test-weak* et *Poids-faute*).

On note *Nb-test-weak-bin* la transformation de *Nb-test-weak* en une variable qualitative de deux catégories (1 si la faute est *weak* au moins pour un test et 0 sinon).

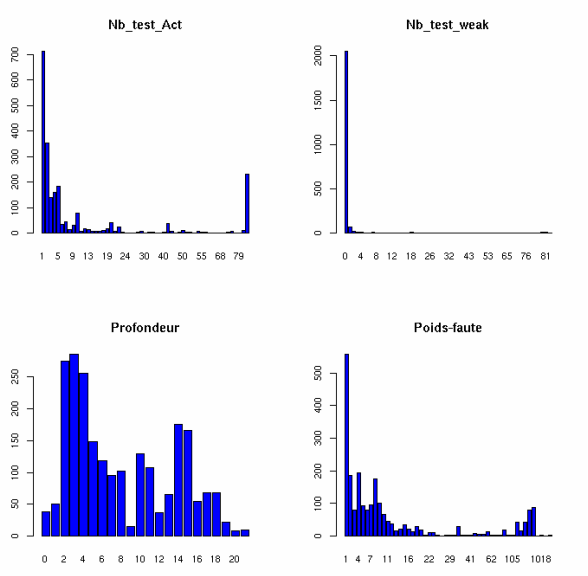

FIG. 28 – Histogrammes des variables quantitatives (*Hc11verilog*)

Adopter des transformations adéquates qui s'appliquent d'une manière automatique pour tous les designs est très important dans cette étude. Ceci est une contrainte qui limite le champ de manipulation des variables. Par exemple on peut améliorer la variable *Nb-test-act* par discrétisation, mais automatiser cette transformation devient très risqué. Dans le cas du compo-

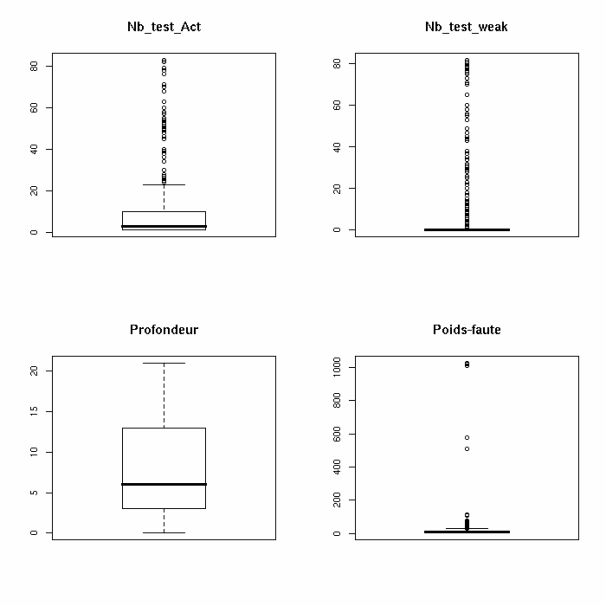

FIG. 29 – Box-plots des variables quantitatives (*Hc11verilog*)

sant *i*8051 on n'a pas besoin de cette opération au moins pour la variable *Nb-test-act* (figure 30, page 45).

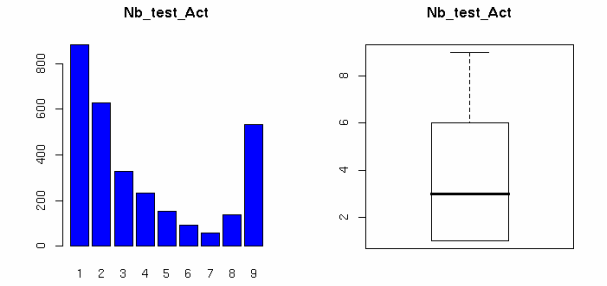

FIG. 30 – Histogramme et box-plot de Nb-Test-Act (I8051)

Une autre contrainte concerne les fautes extrêmes dans la variable *Poids-faute* : éliminer ou garder ces fautes pour une autre analyse particulière est aussi une manipulation délicate loin d'être facile à automatiser.

7.2.2 Odd-ratio et significativité

Type-faute

Le tableau croisé de la variable *Type-faute* avec la variable de réponse *détection* dans le cas du composant $hc11verilog$ est donné par le tableau 13, page 46 :

	non détectée	détectée	Total
Control flow	69	1225	1294
	0.053	0.947	1
Autres fautes	172	831	1003
	0.171	0.828	1
Total	241	2056	2297

TAB. 13 – table croisé de *Type-faute* et détection ($hc11verilog$)

La distance à l'indépendance donnée par le test de χ^2 est significative ici entre type de la faute et détection. Pour donner une interprétation de cette dépendance nous avons calculé la statistique du odds-ratio (rapport des côtes), qui étudie la répartition des *Autres fautes* par rapport aux *Control flow*. Le odd-ratio est égal à $(0.828/0.171)/(0.947/0.053) = 0.272$. Il est bien sûr différent de un (cas de l'indépendance).

Le tableau ci-dessous (tableau 14, page 46) résume cette information dans les quatre design :

Design	χ^2	p-valeur	odd-ratio
$Hc11verilog$	82.75	< 0.001	0.272
$I8051$	56.56	< 0.001	0.212
$router2$	0.818	0.366	1.43
$Edac$	14.85	< 0.001	1.575

TAB. 14 – Test de χ^2 et odd-ratio de la variable *type-faute*

Nb-test-weak-bin

Une analyse analogue a été menée sur la variable transformée *Nb-test-weak-bin*. Alors, on obtient les résultats suivants (tableau 15, page 47) :

Design	χ^2	p-valeur	odd-ratio
$Hc11verilog$	12.55	< 0.001	0.513
$I8051$	23.38	< 0.001	0.303
$router2$	0.24	0.623	1.454
$Edac$	16.33	< 0.001	19.73

TAB. 15 – Test de χ^2 et odd-ratio (*Nb-test-weak-bin*)

Remarque : La fréquence des fautes *weak* par aucun test (*Nb-test-weak-bin* = 0) est très grande dans tous les designs($\geq 88\%$).

Les deux variables qualitatives ont le même comportement dans les composants $hc11verilog$ et $I8051$, où les fautes détectées sont plutôt de type *control-flow* et non *weak* par les tests. Cet effet n'est pas significatif dans le design $router2$.

Profondeur

Les diagrammes en bâton (figure 31, page 47) nous montrent que les fautes profondes sont les plus détectées dans les composants *hc11verilog* et *i8051*.

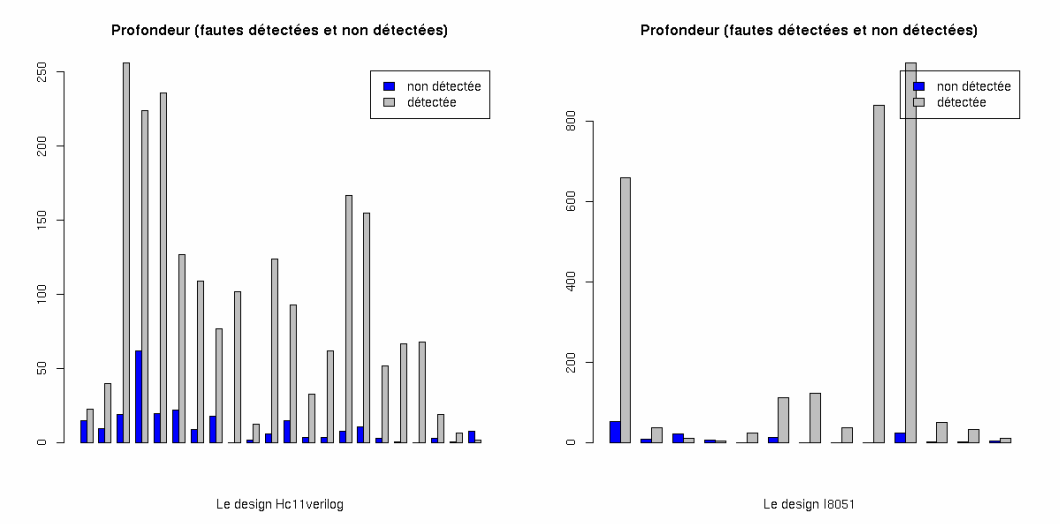

FIG. 31 – Diagrammes en bâton de *Profondeur* dans les deux designs

Poids-faute

La figure 32, page 48 représente la variable *poids-faute* avec les fautes détectés et non détectées. Les fautes avec un poids extrême sont les plus détectées.

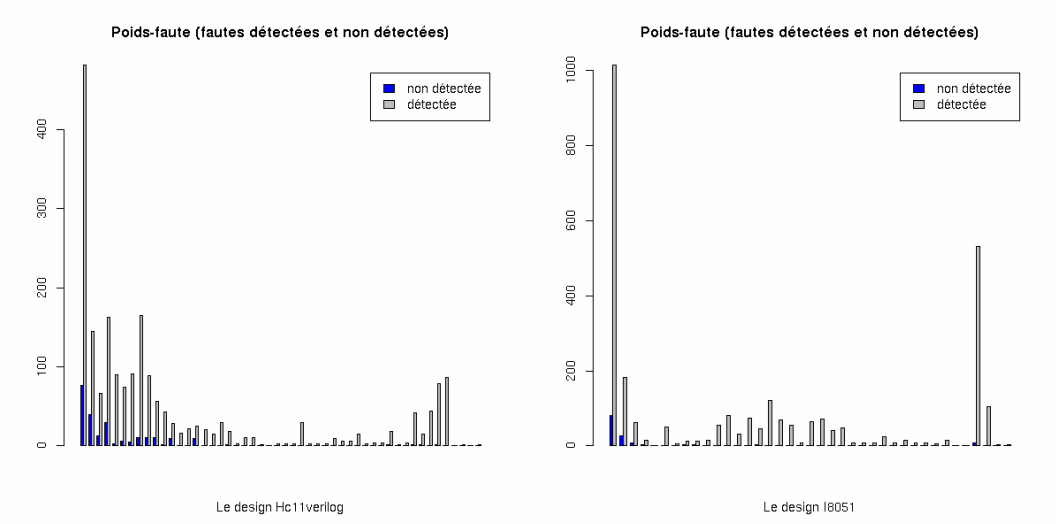

FIG. 32 – Diagrammes en bâton de *Poids-faute* dans les deux designs

L'observation des fautes à poids forts (≥ 510) dans le composant *hc11verilog* et (≥ 176) pour le composant *i8051* donne l'idée de les éliminer de la construction du modèle (poids extrêmement forts). De plus, la plupart de ces fautes sont détectées (plus de 98%) et leur nombre est élevé (219 pour le Composant *hc11verilog* et 672 pour le Composant *i8051*).

Considérer ces fautes comme détectées ou avec une grande probabilité de l'être, nous fait gagner beaucoup de temps. Cette hypothèse est non confirmée, car on risque de ne pas observer le même phénomène dans tous les designs.

7.2.3 Matrice des corrélations

Le tableau ci-dessous (tableau 16, page 49) représente les corrélations des variables quantitatives dans toute la population, les fautes qui ont un poids extrême, et la population en éliminant les fautes extrêmes.

		$Hc11verilog$			$i8051$	
Corr.	Pop.	Extr.	Pop.- Extr.	Pop.	Extr.	Pop.-Extr.
(X_4,X_5)	-0.106	0.335	-0.191	-0.383	0.911	0.006
(X_2,X_5)	0.036	0.447	-0.059	0.082	-0.208	-0.071
(X_2,X_4)	-0.371	0.298	-0.407	-0.405	-0.226	-0.431

TAB. 16 – Corrélations des variables

Une corrélation moyenne entre *Nb-test-act*(X_2) et $profondeur$ (X_4). L'élimination des fautes extrêmes de la variable *poids-faute* (X_5) élimine la partie dépendante entre cette variable et $profondeur$ dans le design $i8051$.

La variable *poids-faute* et *Type-faute* sont corrélées par construction (voir la première partie, section autres définitions).

7.3 Application sur le design $hc11verilog$

Après l'étude univariée et bivariée sur toute la population des fautes, dans la suite, l'estimation du modèle de régression logistique sera sur un échantillon d'apprentissage de taille 700 tiré au hasard de Ω moins les fautes extrêmes (dans la variable *poids-faute*). Par contre pour la métrique, on calcule la prévision sur Ω.

7.3.1 Modèles univariés

La proportion des fautes détectées dans l'échantillon d'apprentissage est $p = 0.8885$.

On commence par les modèles de régression simple avec une variable explicative. Le croisement des variables qualitatives avec la *détection* est présenté dans les tableaux ci-dessous (tableau 17, page 49) pour vérifier qu'il n'y a pas de case de fréquense faible.

	non détectée	détectée	Total
Control flow	20	343	363
Autres fautes	58	279	337
Total	78	622	700

TAB. 17 – table croisée de *Type-faute* et *détection*

Un modèle à une variable explicative X s'ecrit sous la forme suivante :

	non détectée	détectée	Total
0	65	557	622
1	13	65	78
Total	78	622	700

TAB. 18 – table croisée de *Nb-test-weak-bin* et *détection*

$$Logit(P\left[y=1 \mid x\right]) = Ln\left[\frac{P\left[y=1 \mid x\right]}{1-P\left[y=1 \mid x\right]}\right] = \alpha + \beta\, x$$

Le tableau 19, page 50 résume les informations obtenues par l'utilisation d'un programme de la régression logistique sur chacune des co-variables retenues. La significativité du coefficient β est obtenue par le test de rapport de vraisemblance (RV). L_0 est la vraisemblance du modèle nul ($\beta = 0$) et L_1 est la vraisemblance du modèle avec β.

$$RV \;=\; -2ln\frac{L_0}{L_1} \longmapsto \chi^2(1)$$

Dans notre étude, Un paramètre est considéré significatif si la p-valeur correspondante est inférieure ou égale au seuil 0.2.

	$\widehat{\alpha}$	$\widehat{\beta}$	p-valeur	AIC
Type-faute$(X1)$	2.84	-1.27	< 0.001	468.33
Nb-test-act $(X2)$	2.10	-0.0017	0.69	493.13
Nb-test-weak-bin $(X3)$	2.148	-0.538	0.10	490.86
profondeur $(X4)$	1.33	0.105	< 0.001	474.62
poids-faute $(X5)$	0.024	0.013	0.017	487.94

TAB. 19 – La regression logistique sur chacune des co-variables

Parmi les variables dépendantes on garde celles qui affichent avec une p-valeur inférieure au égale à 0.2. Par suite, *Nb-test-act* ne participera pas aux études présentées. Ce critère n'est pas significatif dans la prévision de la détection (au moins pour le composant $hc11verilog$).

7.3.2 Algorithme de choix de modèles

Le nombre de modèles que l'on peut formuler à partir des quatre co-variables (sans interactions) est $2^4 - 1 = 15$ dont les modèles univariés.

On calcul le critère AIC pour chaque modèle et on retient les modèles où le AIC est minimum. Dans un premier temps on ne prend pas en compte les interactions (ableau 20, page 51).

	Modèle avec	AIC
M_1	X_1 et X_3	469
M_2	X_1 et X_4	454.47
M_3	X_1 et X_5	470.3
M_4	X_3 et X_4	476.24
M_5	X_3 et X_5	487.93
M_6	X_4 et X_5	468.84
M_7	X_1, X_3 et X_4	456.37
M_8	X_1, X_3 et X_5	471.71
M_9	X_3, X_4 et X_5	455.88
M_{10}	X_3, X_4 et X_5	470.74
M_{11}	X_1, X_3, X_4 et X_5	457.78

TAB. 20 – L'AIC dans les différents modèles

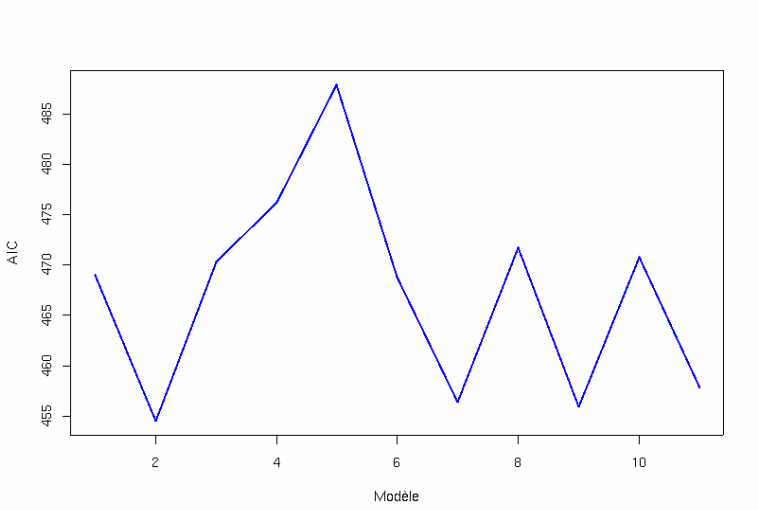

FIG. 33 – L'AIC dans les modèles M_1 jusqu'à M_{11}

L'AIC est minimum pour les modèles M_2, M_7, M_9 et M_{11} (figure 33, page 51) qui ont les modèles condidats. *type-faute* et *profondeur* sont présentes dans tous ces modèles. Leur effet est très important par rapport aux autres critères.
De plus, le modèle M_2 qui contient (*type-faute* et *profondeur*) est le modèle choisi dans une approche automatique descendante à partir du modèle complet. Ce dernire contient toutes les co-variables et toutes les interactions possibles. Cette approche consiste à supprimer un terme à la fois, la composante d'interaction ou l'effet principal qui apparaît comme le moin significatif au sens de l'AIC.

Remarque : Aucune interaction n'est significative ici.

7.3.3 Comparaisont et validités des modèles

Pour finaliser le choix d'un modèle, nous avons utilisé l'erreur de prédiction plutôt que l'ajustement. Nous avons calculé deux estimateurs de cette erreur, La première est obtenue par resubstitution des données d'apprentissage. C'est le taux de mal classés ou taux d'erreur. La deuxième estimation opère par validation croisée (cf. la bibliographie numéro 5).

Le pouvoir discriminant des modèles

A partir des probabilités estimées on peut décider en fixant un seuil, par exemple à 0.85, de classer les fautes dans la catégorie détectées ($y = 1$) si sa probabilité est supérieure au seuil et dans la catégorie non détectées ($y = 0$). Il s'agit d'une règle de classement :

La *sensibilité* est alors définie comme la probabilité de classer la faute dans la catégorie $y = 1$ étant donné qu'elle est effectivement observée dans celle-ci.
La *specificité* est la probabilité de classer la faute dans la catégorie $y = 0$ étant donné qu'elle est effectivement observée dans celle-ci.

Le tableau 21, page 53 ci-dessous montre un exemple de classement obtenu pour un seuil égal à la proportion des fautes détectées (0.888) (modèle M_2) sur les données correspondant au composant $hc11verilog$.

		Observé		
		non détectée	détectée	Total
Classement	y = 0	53	191	244
	y = 1	25	431	456
	Total	78	622	700

TAB. 21 – Table de classement (modèle M_2)

sensibilité $= 431/622 = 0.693$, specificité $= 53/78 = 0.68$

Le *taux d'erreur* est la proportion des fautes mal classées dans la catégorie $y = 1$ dans l'échantillon. Pour le modèle M_2 égale à $25/700 = 0.0357$ (tableau 22, page 53).

Modèle	sensibilité	specificité	taux d'erreur	métrique
M_2	0.693	0.68	0.0357	0.8932
M_7	0.693	0.679	0.0357	0.8932
M_9	0.691	0.653	0.038	0.9000
M_{11}	0.691	0.654	0.038	0.9002

TAB. 22 – sensibilité, specificité, taux d'erreur et métrique

La métrique par définition est la moyenne des probabilités obtenues par le modèle. On rappelle que la métrique réelle est égale à 0.8951.
Pour un seuil égal à 0.888, on a des taux d'erreurs égaux entre le modèle qui contient *type-faute* et *profondeur* (M_2) et celui qui contient en plus *Nb-test-weak-bin* (M_7) (tableau 22, page 53). Dans ce cas, on choisit plutôt le modèle réduit qui contient le moins de variables.

Utilisation de la validation croisée (V.C)

La validation croisée est conceptuellement simple, efficace et largement utilisée pour estimer une erreur quand la taille de l'échantillon initial est réduite. Elle opère en trois étapes :

- Découper aléatoirement l'échantillon en K parties de tailles approximativement égales selon une loi uniforme ;
- répéter K fois l'opération qui consiste à mettre de côté l'une des parties, estimer le modèle sur les $K - 1$ parties restantes ;
- moyenner toutes ces erreurs pour aboutir à l'estimation par validation croisée.

Pour $K = 14$, on estime chaque fois le modèle sur un échantillon de 650 fautes et on calcule le taux d'erreur pour les 50 fautes écartées. De plus, on calcule la moyenne et l'écart-type des métriques obtenues dans chaque étape. Les résultats obtenus sur les données correspondant au composant $hc11verilog$ sont présentés dans le tableau 23, page 54.

Remarque : On garde la même partition de l'échantillon pour tous les modèles.

Modèle	Erreur V.C	moyenne de la métrique	écart-type
M_2	0.0428	0.8926	0.0035
M_7	0.0428	0.8925	0.0035
M_9	0.0457	0.8978	0.0031
M_{11}	0.0442	0.8993	0.0032

TAB. 23 – Erreur de prédiction et métrique par validation croisée

Une erreur de prédiction légèrement élevé des modèles M_9 et M_{11}. Les modèles M_2 et M_7 sont les meilleurs pour prédire la détection. La différence enter ces dèrniers est la présence de la variable *Nb-test-weak-bin* ($X3$) dans le modèle M_7, cela confirme l'effet non significatif de la variable *Nb-test-weak-bin*. On remarque que la moyenne de la métrique estimée par la validation croisée est faible par rapport au calcul normale pour tout les modèles.

7.3.4 Estimation par bootstrap

Le but de cette approche statistique est d'utiliser une technique de rééchantillonnage pour créer un grand nombre (B) d'échantillons par échantillonnage aléatoire avec remise selon la loi uniforme dans l'échantillon initial. On estime un modèle logistique et on calcule la métrique correspondante sur chacun de ces échantillons. On obtient ainsi un vecteur du taille B de la variable métrique, et son écart-type représente l'erreur estimée par le boostrap.

Pour $B = 1000$, on résume les résultats dans le tableau ci-dessous (tableau 24, page 55).

L'erreur de la métrique estimée par la technique de bootstrap sur le modèle logistique M_2 est faible par rapport aux autre modèles.

On peut modéliser la détection dans le composant *hc11verilog* par un modèle de la régression logistique où les variables *type-faute* et *profondeur*

Modèle	moyenne de la métrique	écart-type
M_2	0.8922	0.0108
M_7	0.8937	0.0113
M_9	0.8879	0.0283
M_{11}	0.8909	0.0251

TAB. 24 – estimation par bootstrap

sont les prédicteurs sur un échantillon de 700 fautes tiré au hasard. La métrique estimée égale à 0.8922 avec 0.0108 de présision (écart-type).

7.3.5 Validation par simulations

Jusqu'à maintenant, tout le travail a été fait sur un échantillon d'apprentissage, qui a servi en même temps pour calculer une erreur de prédiction. On peut vérifier et comparer la capacité d'un modèle logistique à estimer la métrique et comment il corrige la proportion des fautes détectées dans l'échantillon, en utilisant l'information complète de la phase de détection :

On connaît exactement la métrique = 0.8951 pour le composant *hc11verilog*. On tire chaque fois un échantillon aléatoire de taille 500, p est la proportion des fautes détectées dans l'échantillon. On calcule la métrique obtenue par un modèle logistique et on observe sa distance avec la métrique réelle et la proportion. Pour le modèle M_2 et 50 échantillons (figure 34, page 56).

Pour comparer entre la proportion et la métrique estimée par un modèle logistique on calcule la moyenne des écarts avec la métrique réelle obtenue par l'étape précidente (tableau 25, page 55).

Modèle	moyenne des écarts (Métrique)	moyenne des écarts (Proportion)
M_2	0.0090	0.00984
M_7	0.0093	0.00984
M_9	0.0091	0.00984
M_{11}	0.0091	0.00984

TAB. 25 – La moyenne des écarts avec la métrique réelle ($hc11verilog$)

La moyenne des écarts entre la métrique estimées par un des quatre modèle logistique et la métrique réelle est faible par rapport a la moyenne des écarts entre la proportion et la métrique réelle pour le composant $hc11verilog$.

On observe presque le même comportement entre les modèles M_2 et M_{11} (figure 35, page 56). Le nombre de tests où la faute est weak (*nb-test-weak-*

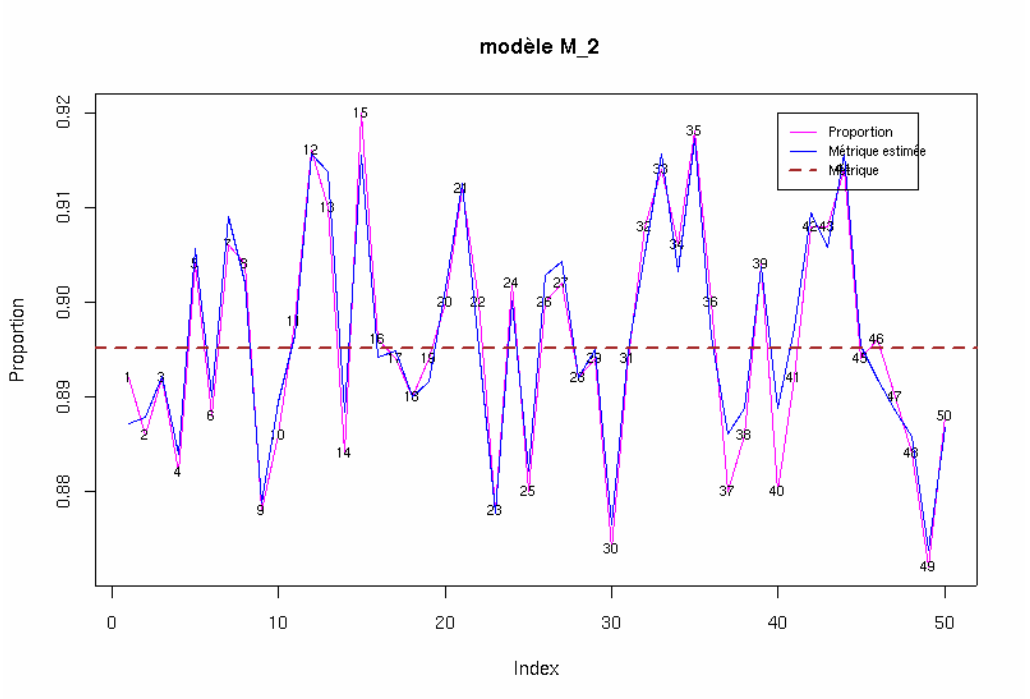

FIG. 34 – Comparaison et vérification modèle M_2

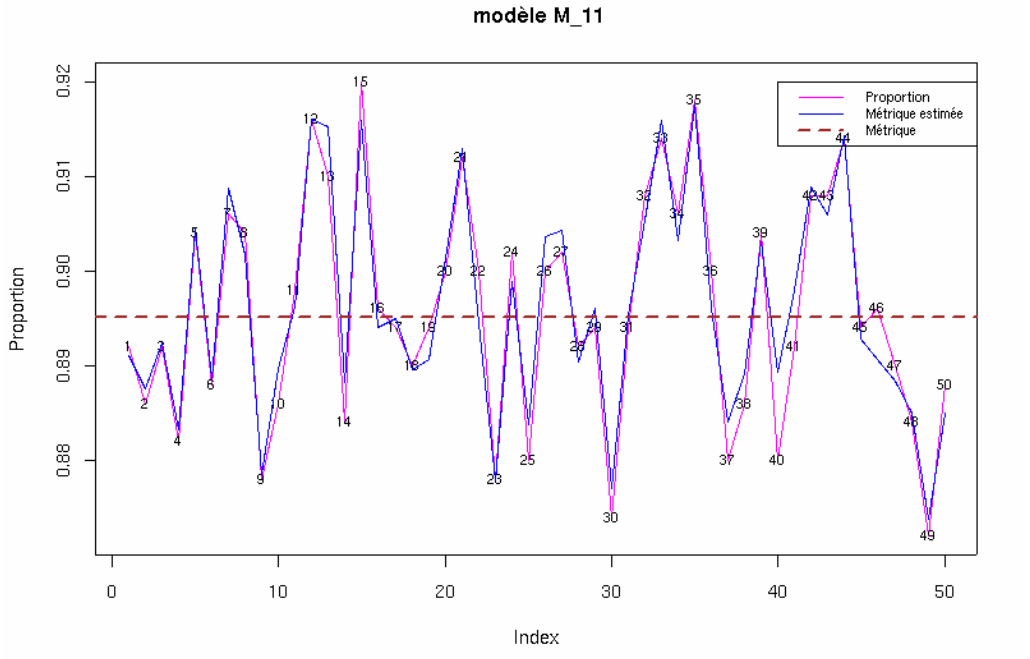

FIG. 35 – Comparaison et vérification modèle M_{11}

bin) et le poids de la faute (*poids-faute*) ont un effet faible dans la prédiction de la métrique pour le $hc11verilog$.

7.4 Application sur le design $i8051$

La proportion des fautes détectées dans l'échantillon d'apprentissage de taille 700 tiré au hasard de Ω moins les fautes extrêmes est $p = 0.941$ pour le composant $i8051$.

Le même travail a été fait pour le $i8051$. On résume dans le tableau 26, page 57 les résultats de la régression logistique sur chaque variables :

	$\widehat{\alpha}$	$\widehat{\beta}$	p-valeur	AIC
Type-faute$(X1)$	4.204	-1.847	< 0.001	297.57
Nb-test-act $(X2)$	4.041	-0.276	< 0.001	288.39
Nb-test-weak-bin $(X3)$	2.835	-1.4485	0.029	312.64
profondeur $(X4)$	0.727	0.342	< 0.001	257.29
poids-faute $(X5)$	2.122	0.165	0.0013	290.22

TAB. 26 – La régression logistique sur chacune des co-variables

Les modèles univariés sont significatifs pour prédire la détection. La variable *Nb-test-act* (à l'inverse du cas de composant *hc11verilog*) rentre dans la recherche d'un modèle multivariés.
Le modèle choisi par une procédure descendante à partir du modèle complet sans interactions est celui qui contient les variables *Type-faute*, *profondeur* et *poids-faute* avec AIC = 239.47.

On peut aussi proposer d'autres modèles concurrents après observation de l'AIC, en gardant les variables *profondeur* (X_4) et *poids-faute* (X_5) (tableau 27, page 57) :

	Modèle avec	AIC
M_1	X_1, X_3, X_4 et X_5	239.54
M_2	X_1, X_2, X_4 et X_5	240.31
M_3	X_1, X_4 et X_5	239.47
M_4	X_4 et X_5	240.80

TAB. 27 – L'AIC dans les différents modèles

Taux d'erreur et métrique

Pour un seuil égal à 0.941 on calcule les sensibilités, specificités, taux d'erreurs et métriques (tableau 28, page 58) :

Modèle	sensibilité	specificité	taux d'erreur	métrique
M_1	0.830	0.756	0.014	0.9560
M_2	0.836	0.683	0.018	0.9566
M_3	0.836	0.683	0.018	0.9566
M_4	0.850	0.683	0.019	0.9567

TAB. 28 – sensibilité, specificité, taux d'erreur et métrique

Le modèle M_1 qui contient *Nb-test-weak-bin* a le taux d'erreur le moins élevé et une spécificité supérieure (tableau 29, page 58). Un effet très faible de *Nb-test-act* dans le modèle M_2, qui explique le même comportement des modèles M_2 et M_3.

Validation croisée (V.C)

L'estimation par validation croisée ne montre pas une grande différence entre les modèles condidats. Cependant, le modèle M_1 est légèrement mieux (tableau 29, page 58).

Modèle	Erreur V.C	moyenne de métrique	écart-type
M_1	0.0242	0.957	0.0019
M_2	0.0285	0.957	0.0018
M_3	0.0285	0.957	0.0018
M_4	0.0286	0.957	0.0018

TAB. 29 – Erreur de prédiction et métrique par validation croisée

Dans le $i8051$, la profondeur et le poids des fautes sont les critères les plus pertinants.

Estimation par bootstrap

Pour B égal à 1000 on trouve les résultats suivants (tableau 30, page 59) :
Les modèles ont un comportement presque similaire dans l'estimation de la marge d'erreur de la métrique.

Modèle	moyenne de la métrique	écart-type
M_1	0.9557	0.0064
M_2	0.9562	0.0059
M_3	0.9566	0.0061
M_4	0.9565	0.0061

TAB. 30 – Estimation par bootstrap i8051

Un modèle logistique avec les prédicteurs *Type-faute*, *profondeur* et *poids-faute* nous permet de modéliser la détection, d'estimer la métrique du composant $i8051$ égale à 0.9566 avec une erreur de prédiction égale à 0.0285 et écart-type de la métrique obtenue par la technique de bootstrap égale à 0.0061.

Validation par simulations

On observe la métrique obtenue par la régression logistique dans les modèles M_1 et M_3 et la proportion des fautes détectées dans chaque échantillon. Pour 50 échantillons du taille 500 les résultats sont représentés dans la figure 36, page 60).

On voit dans les figures ci-dessus que la métrique estimée par les modèles M_1 et M_3 corrige la proportion des fautes détectées (en rouge), surtout dans l'échantillon 8, 15, 29, 35, 36 et 41 où la proportion est très loin de la métrique réelle.

Modèle	moyenne des écarts (Métrique)	moyenne des écarts (Proportion)
M_1	0.00681	0.00795
M_2	0.00663	0.00795
M_3	0.00669	0.00795
M_4	0.00665	0.00795

TAB. 31 – L'écart avec la métrique réelle ($i8051$)

La métrique estimée par un modèle logistique est en moyenne plus proche de la vraie valeur que la proportion.

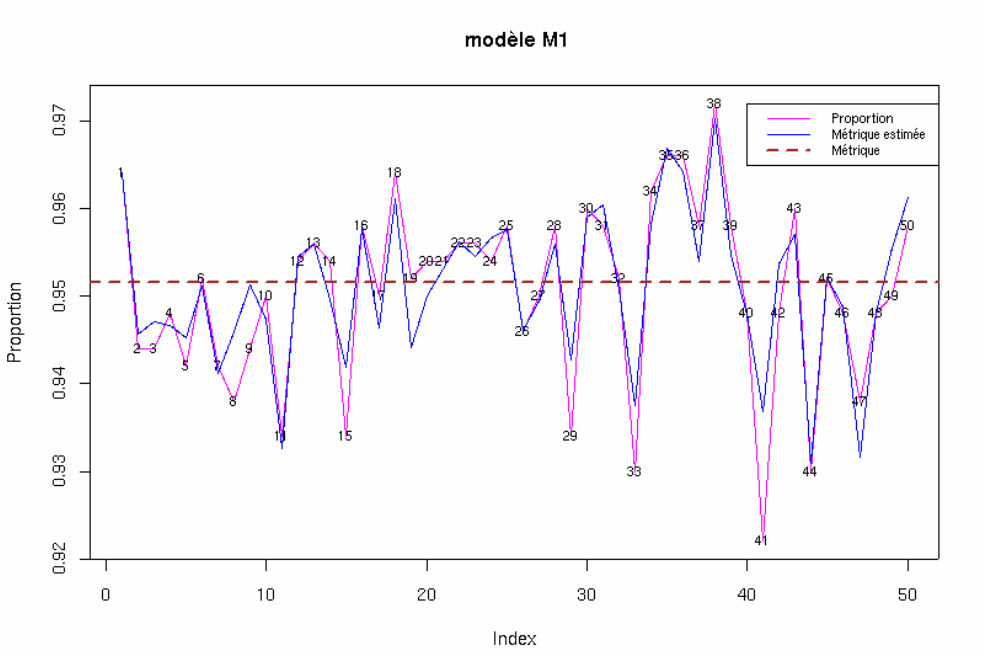

FIG. 36 – Comparaison et vérification modèle M_1

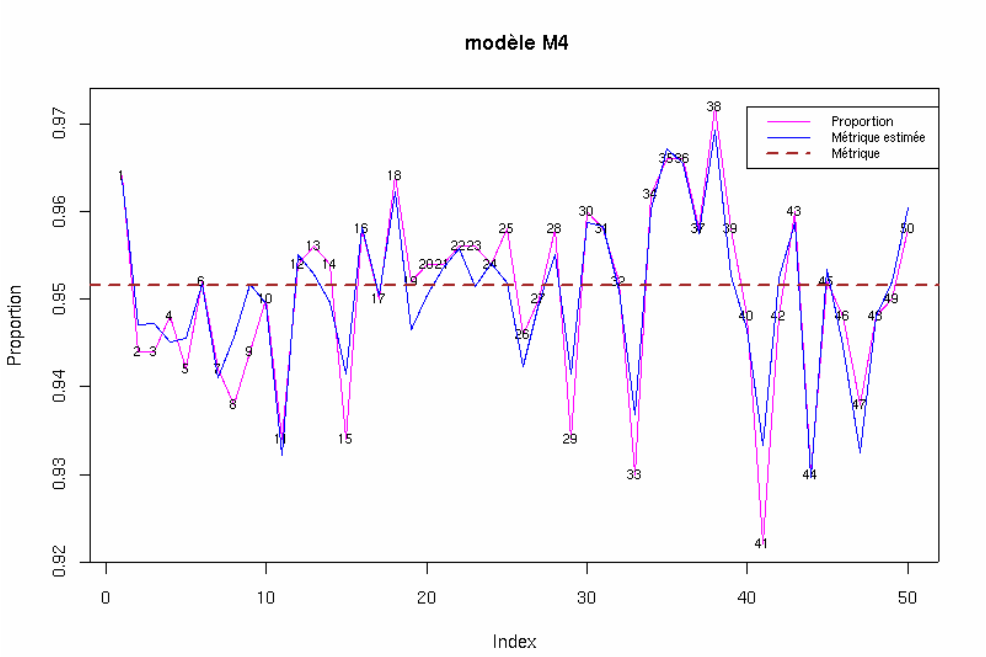

FIG. 37 – Comparaison et vérification modèle M_3

7.5 Limites et avantages

L'estimation de la métrique par la régression logistique a des limites et des avantages.

Les limites

- La différence entre designs ne permet pas de définir un modèle général.
- Le principe de l'estimation est basé sur la prévision, donc avoir un échantillon représentatif est très important.
- Dans le cas des petits échantillons et de taux de détection important, le risque d'avoir des profils mal présentés (effectif faible) et très élevé.

Les avantages

- Fournir la probabilité qu'une faute est détectée conditionnellement à des prédicteurs.
- Déterminer les critères pertinants pour prévoir la détection.
- Rassembler plusieurs prédicteurs dans un ou plusieurs modèles.
- Estimer l'erreur de prédiction, la métrique et la marge d'erreur.

Conclusion

Ce stage s'est déroulé dans une société spécialisée dans l'E.D.A, un domaine que je ne connaissais pas auparavant. L'objectif final de cette étude tourne autour de la *métrique*. Les analyses et les méthodes statistiques appliquées au cours de ce stage ont permis d'avoir une vision claire sur le processus de test d'un circuit électronique et de proposer des solutions au problème initial.

L'étude de similarité en utilisant les résultats de la phase d'activation permet de mesurer la liaison entre les tests. L'ACM nous a montré la différence entre les designs étudiés et la pertinence de certains critères (*firsttime*). Ces variables mériteraient une étude plus approfondie.

Estimer la probabilité qu'une faute soit détectée est l'une des solutions proposées, à l'aide d'un modèle logistique construit à la base des connaissances préalables disponibles pour chaque faute.

Ce stage a été pour moi un défi, parce qu'il a demandé beaucoup d'autonomie dans le travail à cause de l'absence des statisticiens expérimentés dans la société *Certess*. Mais ce stage m'a permis d'essayer de trouver des solutions à des problèmes réels. Cependant, j'ai rencontré des difficultés pendant la réalisation de cette étude du fait de manque du repère sur le plan théorique sur lesquel j'aurais pu me baser .

Bibliographie

[1] SAPORTA G. *Probabilités analyse des données et statistique*, édition TECHNIP, 1990.

[2] Lebart L, Morineau A et Piron M. *Statistique exploratoire multidimensionnelle*, édition DUNOD, 2004.

[3] Crucianu M, Asselin de Beauville JP et Boné R. *Méthodes factorielles pour l'analyse des données*, édition LAVOISIER, 2004.

[4] Sous la direction de Govaert G. *Analyse des données*, édition LAVOISIER, 2003.

[5] Besse P. *Data mining (modélisation statistique et apprentissage)*. Publications du laboratoire de statistique et probabilités (Toulouse III), Mars 2005. www.lsp.ups-tlse.fr/Besse.

[6] Desjardins J. (Université de Montréal). *L'analyse de la régression logistique* (Tutorial in quantitative methods for psychology). Septembre 2005. http ://www.tqmp.org/doc/vol1-1/p35.pdf .

[7] Taffé P. *Cours de régression logistique appliquée*. Institut universitaire de médcine sociale et préventive (IUMSP) et centre d'épidémiologie clinique (CepiC), Lausanne, Août 2004.
http ://www.tesser-pro.org/stat/Cours_regression_logistique.pdf.