AHMED AROUS

STRATÉGIES DE LOCALISATION DU (DES) COMPOSANT(S) DÉFAILLANT(S) POUR UN SYSTÈME MULTI-COMPOSANT

Mémoire présenté
à la Faculté des études supérieures et postdoctorales de l'Université Laval
dans le cadre du programme de maîtrise mémoire en génie mécanique
pour l'obtention du grade de Maître ès sciences (M.Sc.)

DÉPARTEMENT DE GÉNIE MÉCANIQUE
FACULTÉ DES SCIENCES ET GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2012

Résumé

Dans ce mémoire nous traitons le problème de localisation du (des) composant(s) responsable(s) de la défaillance. Chaque composant est assujetti à des défaillances aléatoires. La détection de l'état d'un composant ou d'un sous-système est effectuée à l'aide de tests. L'objectif de cette recherche est d'exploiter les techniques et connaissances disponibles pour générer la séquence de tests qui permet de localiser rapidement le(s) composant(s) responsable(s) de la défaillance du système. On considère un système opérant suivant une structure série pour lequel on connaît le coût de tests et la probabilité conditionnelle qu'un composant (i) soit responsable de la défaillance. On analyse les différentes stratégies de diagnostic. Des exemples, empruntés à la littérature, sont utilisés pour illustrer chaque procédure traitée. Des extensions sont proposées pour traiter le cas où le diagramme de fonctionnement du système n'est pas nécessairement "série". Les algorithmes traités font appel à l'analyse probabiliste des systèmes, à la théorie de l'information, à l'approche heuristique et à la programmation dynamique.

Abstract

In this paper, we address the problem of the localization of the component(s) responsible(s) for the failure. Each component is subjected to random failures. Some tests help the detection of the state of a component or a subsystem. The objective of this research is to exploit the available knowledge and techniques to generate the tests sequence that locate quickly the (s) component (s) responsible (s) of system failure. We consider a system which operates according to a structure series and of which we know the test costs and the conditional probability that a component (i) is out of service. We analyze the different diagnostic strategies. Some examples, taken from the literature, are used to illustrate each procedure covered. Many extensions are proposed to handle the case where the diagram of the system is not necessarily "series". The algorithms treated are based of probabilistic analysis of systems, the information theory, the heuristic approach and the dynamic programming.

Dédicaces

À mes chers parents ***Farhat*** *et* ***Rafikha***

À qui je dois les meilleurs de moi-même en témoignage de ma profonde affection et de ma sincère reconnaissance pour les grands sacrifices déployés à mon égard. Qu'ils veuillent bien, ceux qui n'attendaient que ce jour-là, trouver dans ce travail une modeste reconnaissance à leurs sacrifices.

À mon cher frère ***Issa*** *et ma chère sœur* ***Myriam***

Nulle dédicace ne saurait témoigner de l'étendue des sentiments que j'éprouve à leur égard pour l'amour, le soutien et le goût de faire plus et mieux dont elles ont su m'entourer.

À mes chers amis

Pour tant d'amour et d'affection pour m'avoir aimé, suivis, aidés et encouragés.

À mon encadreur ***Daoud Ait-Kadi***

Pour sa disponibilité…

Pour son assistance, son soutien et ses encouragements tout au long de ma maitrise

À tous ceux qui me sont chers

Ahmed Arous

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1 : Introduction

1. Introduction

Les systèmes industriels sont de plus en plus complexes. Ils utilisent plusieurs technologies et composants dont les interactions sont parfois difficiles à modéliser et à prévoir. Il en résulte qu'en cas de défaillance du système, il n'est pas toujours simple de localiser le (les) composant(s) responsable(s) de la défaillance.

Le technicien de maintenance va souvent faire appel à son expérience pour la détection de la panne. Ils proposent des procédures d'aide au diagnostic et de maintenance. Ces procédures ne sont pas toujours simples à effectuer vu que l'application de maintenance exige des délais courts pour une efficacité maximale.

Dans un contexte où l'état d'un composant n'est connu qu'après inspection, il est important de mettre en œuvre une stratégie de localisation qui permet d'identifier le composant défectueux. Divers travaux ont été publiés sur le problème de localisation rapide. Ces travaux tentent d'exploiter l'information disponible pour générer la séquence de tests à effectuer. Les tests prennent fin une fois que le composant responsable de la défaillance est détecté. L'information relative à un composant peut être connue, partiellement connue ou totalement inconnue. La séquence de tests peut être séquentielle ou non séquentielle suivant le domaine d'application et les limites qu'on dispose. Chaque composant peut être soit en bon état ou hors d'usage. En absence de redondance, la défaillance d'un composant entraîne la défaillance du système. Pour être dans le contexte, une petite introduction aux différentes étapes de la maintenance corrective sera présentée dans la section 2.

2. Les étapes de la maintenance corrective

La maintenance corrective désigne « l'élimination d'une avarie ou d'une altération dans le fonctionnement d'un élément matériel par un des divers moyens que sont la réparation, la restauration à l'état antérieur, et le remplacement de l'élément matériel impliqué» [28]. Une fois la panne est déclarée, chaque technicien de maintenance doit croiser les étapes suivantes :

2.1. La détection de la panne

La détection de la panne est la phase où la défaillance est mise en évidence. Elle peut se produire soit d'une façon visuelle ou automatique.

2.2. Le diagnostic de la panne

De son coté, le diagnostic est la réponse aux questions suivantes :

- Quel est le composant défaillant?
- Quelle est la méthodologie de détections de pannes que l'on doit appliquer relativement aux données qu'on dispose ?

Dans un premier stade de diagnostic, la panne doit être repérée et ensuite, une explication des causes à l'origine doit être associée. Dans ce mémoire, on s'intéresse plutôt à la localisation de la défaillance.

2.3. La réparation de la panne

Une fois la méthode de diagnostic est mise en place, on exécute les tests jusqu'à ce que la panne soit repérée. Enfin, on doit remplacer le composant fautif et tester la machine.

Remarque

La phase de détection de la panne est une phase pratiquement instantanée. De son coté, l'étape de réparation possède une durée constante. Cette durée dépend généralement de la technologie mise en place et des équipements disponibles. Par conséquent, l'étape de diagnostic constitue la phase critique. La durée relative à cette phase peut être diminuée afin d'augmenter la disponibilité de la machine. L'efficacité et la rapidité de la phase de diagnostic sont les objectifs recherchés dans ce mémoire.

3. Contenu du mémoire

Le contenu du mémoire est réparti sur six chapitres :

La problématique est décrite dans le premier chapitre. Une introduction générale au diagnostic de pannes et aux étapes de la maintenance corrective est présentée.

Le chapitre 2 porte sur les stratégies de diagnostic. Des exemples empruntés à la littérature sont traités pour expliquer chaque stratégie.

Le troisième chapitre est consacré à la génération de la séquence optimale de tests. Une approche probabiliste sera présentée. Cette approche se base sur les probabilités conditionnelles de défaillance et les coûts encourus pour tester chaque composant formant le système. Les données utilisées dans les modèles générés sont connues.

La théorie d'information, combinée à la programmation dynamique et l'approche heuristique, fera l'objet du quatrième chapitre. Les modèles proposés se basent sur un dictionnaire de tests binaires reliant le résultat du test à la cause de défaillance. Les algorithmes construits peuvent être séquentiels ou non séquentiels suivant les limites imposées.

Dans le chapitre 5, on traite des stratégies de localisation de défauts pour des systèmes complexes où plusieurs composants peuvent tomber en pannes simultanément.

Une conclusion des travaux réalisés dans le cadre de cette recherche est présentée à la fin de ce mémoire. On propose aussi les perspectives et approfondissements ouverts par ce mémoire. Une liste de références bibliographiques est également fournie.

Chapitre 2 : Les stratégies de diagnostic et de localisation de défauts

1. Introduction

CHANDRASEKARA et MILNE [5] considèrent plusieurs niveaux de traitement d'un système de diagnostic. Ces niveaux sont souvent interconnectés et constituent les informations de base du système. Les niveaux utilisés sont, outre la structure du système, son comportement, ses fonctions et ses modes de défaillances. Le service de maintenance saisit l'information et suivant la disponibilité des connaissances, il commence le diagnostic du niveau le moins difficile. Le passage d'un niveau à un autre dépend du flux d'information et de l'utilité des données disponibles à ce niveau. Pour la souplesse du travail, plusieurs chercheurs essaient de créer des règles qui relient d'une part les fonctions aux modes de défaillances ou bien les mesures observées aux fonctions d'un système…

FELLOUAH [7] présente quelques critères de performance pour un diagnostic rapide. Parmi ces critères, on retrouve:

- La détectabilité;
- La sensibilité;
- La robustesse;
- Le coût;
- Le temps réel de diagnostic;
- La probabilité de défaillance conditionnelle du composant.

La rapidité et la performance d'un système de diagnostic sont fonction des critères considérés. Ces critères sont choisis suivant l'information disponible et la nature du système.

2. Les stratégies de diagnostic

2.1. La reconnaissance de formes

La reconnaissance de formes est « un ensemble de techniques et méthodes visant à identifier des motifs à partir des données brutes dans l'intention de prendre une décision dépendante de la catégorie attribuée à ce motif » [28].
Le raisonnement à partir des cas fait partie des techniques de reconnaissance de formes. C'est un réflexe qui « résout les problèmes en retrouvant des cas analogues dans la base de connaissances et en les adaptant au cas considéré » [17].
Cette méthodologie s'appuie sur un ensemble de règles théoriques. Comme montre la définition, elle explique à quel type, une forme observée ressemble le plus. Le diagnostic est établi à partir d'une analyse de similarité des cas. On utilise un problème (cas) déjà résolu afin de traiter les problèmes actuels. Ensuite, chaque cas résolu est mémorisé pour enrichir la base de connaissance qui sera utilisée dans les problèmes futurs (voir section 2.2).

2.2. Le datamining

Le datamining ou l'extraction des connaissances à partir des données (ECD) est un domaine bien fréquenté. Il est utilisé pour l'exploitation des données et la capitalisation du savoir-faire. On définit le datamining comme un processus non trivial d'identification de structures inconnues, valides et potentiellement exploitables dans les bases de données.

MEKROUD et MOUSSAOUI [17] ont combiné les techniques du datamining et celle du raisonnement à partir des cas pour la détection de pannes dans une extrudeuse de tubes en plastique. Ces deux méthodes émergent pour la mise en cause de la liaison directe entre les différentes pannes et leurs solutions possibles à partir des techniques de gestion des connaissances.

Le travail de MEKROUD et MOUSSAOUI [17] consiste à fragmenter la base des cas en descripteurs-pannes et descripteurs-solutions. Ensuite, il faut classifier les données correspondant à chaque espace suivi d'un mappage entre les clusters des deux fragments à travers les mesures de similarités. Si une panne se déclare, il faut préparer le code (les descripteurs) de ce nouveau cas pour chercher, dans l'espace des descripteurs-pannes, le cas le plus proche en utilisant un cycle RàPC (raisonnement à partir de cas). Lorsque le problème similaire est trouvé, il faut mapper vers le cluster pertinent dans l'espace des solutions suivies d'un deuxième cycle RàPC pour trouver la solution appropriée du cas à résoudre.
Pour ce travail, une base de données avec des problèmes déjà résolus doit être disponible. Sinon, si une nouvelle panne se déclare, cette approche devient inefficace.

2.3. La logique floue

La logique floue est une approche qui combine la modélisation numérique et la modélisation symbolique. Elle est exploitée couramment dans le domaine d'aide à la décision vu sa simplicité de mise en œuvre. Pour une information incertaine ou erronée qu'on peut obtenir lors d'une procédure de diagnostic, la logique floue entre en jeu pour traiter les données imprécises et imparfaites qui seront traduites sous forme de règles floues (voir section 2.4).

2.4. Les réseaux de neurones

D'après HAYKIN [14], un réseau de neurones est « un processus massivement distribué en parallèle qui a une propension naturelle pour stocker de la connaissance empirique et la rendre disponible à l'usage ». Selon BARTHELEMY [2], cette technique est fondée sur la ressemblance faite avec le cerveau humain à travers les deux perspectives suivantes :

- Le processus d'apprentissage acquis par la connaissance humaine.
- Les poids synaptiques qui reflètent les connexions entre les neurones et servent à stoker les connaissances.

Cette ressemblance a attiré quelques chercheurs pour croire à l'intérêt que peut rapporter la mise en évidence des réseaux de neurones dans le problème de diagnostic des pannes. Cette technique de type « boîte noire » est utilisée fréquemment dans le cas où la connaissance est absente à travers les techniques d'apprentissage par retour à l'expérience.

MAHDAOUI et MOUSS [16] ont combiné les techniques des réseaux de neurones avec celle de la logique floue pour le diagnostic d'un système de production. Chaque technique possède ses propriétés et arrive à résoudre un certain type de problèmes. Ces limites d'utilisation ont été le point de départ pour la création des systèmes neuro-flous. Sachant que, dans certains cas, l'observation d'un mode de défaillance n'est pas évidente. Ainsi, le travail de MAHDAOUI et MOUSS [16] entre dans la phase d'identification du mode de défaillance. Ces chercheurs ont défini le logiciel NEFDIAG pour classifier les modes de défaillance. En présence des capteurs, NEFDIAG [16] construit son système d'apprentissage par un ensemble de formes, puis chaque forme observée est classée dans une des classes prédéfinies. Ce logiciel produit des règles floues par un parcours de données et optimise ensuite les règles par apprentissage des sous-ensembles floues qui sont utilisées pour partitionner les données [16]. NEFDIAG fonctionne de la façon suivante :

Si

Variable1 est B_1
Variable2 est B_2
Variable3 est B_3
Variable n est B_n

Alors, la forme $(x_1, x_2, x_3, \ldots, x_n)$ est classée dans la classe « mode de défaillance 1 ».

2.5. Les méthodes déductives

Les méthodes de diagnostic déductives suivent une hiérarchie descendante. En effet, on part de l'événement indésirable pour atteindre les causes possibles.

2.5.1. L'AMDEC

2.5.1.1. Définition

L'AMDEC (**A**nalyse des **M**odes de **D**éfaillance, de leurs **E**ffets et de leur **C**riticité) permet de générer une base d'information puissante pour définir, améliorer, corriger et valider une conception, un procédé ou un moyen, tout au long de la vie du produit, depuis la conception jusqu'à sa fin de vie [25] (voir Figure 1). Cette méthode illustre tous les modes de défaillance, leurs relations avec les fonctions analysées, les symptômes que peuvent engendrer et les causes qui sont à l'origine.

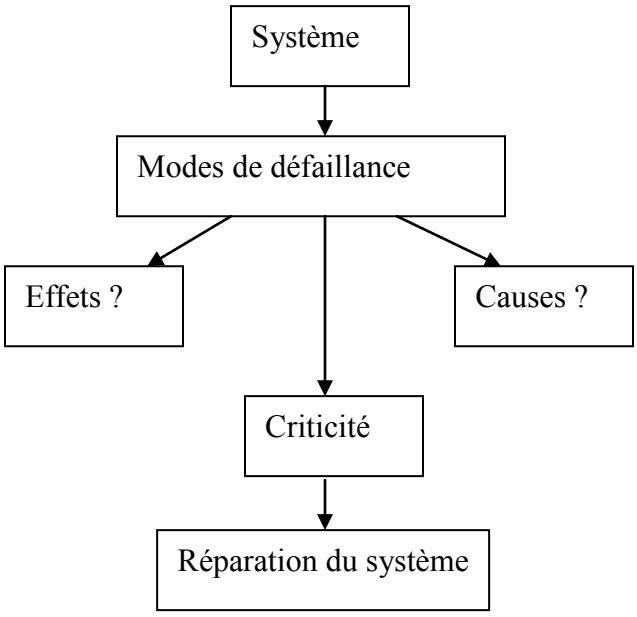

Figure 1. Définition de l'AMDEC

2.5.1.2. Contribution AMDEC-diagnostic

Concernant l'AMDEC, l'analyse qualitative correspond à l'analyse des modes de défaillances et de leurs effets alors que cette analyse devient quantitative lorsqu'une attention sera portée à la criticité.

- *L'analyse quantitative*

L'AMDEC une fois effectuée, elle fournit une base d'information assez puissante pour un diagnostic rapide. Devant un mode de défaillance, le technicien doit revenir aux résultats de l'AMDEC pour raisonner, partant des symptômes observés jusqu'aux causes responsables.

Plusieurs chercheurs ont exploité l'AMDEC pour le diagnostic des défaillances. EMOND [6] a utilisé les connaissances issues de cette analyse pour le suivi des pannes. Ces connaissances sont généralement construites à partir de propositions logiques qui relient les défaillances aux symptômes possibles. De son coté, GLOUD [13] a utilisé une heuristique qui relie les fonctions aux modes de défaillance. Il sera plus judicieux par contre d'utiliser les résultats de l'AMDEC reliant les causes aux modes de défaillances. Ces relations seront capturées dans une représentation unique et puis analysées. On propose un exemple très simple :

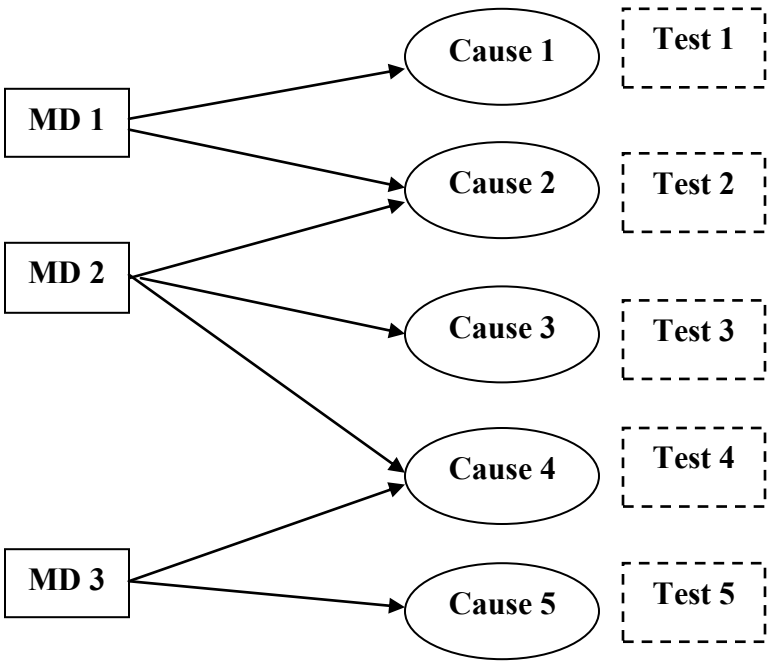

Figure 2 .Relations entre les modes de défaillances et leurs causes

Trois modes de défaillance peuvent être observées. La défaillance du système peut être due à cinq causes dont chacune est détectée par un seul test (voir Figue 2). On propose les deux règles suivantes :

Règle 1 :

Si un mode de défaillance est observé, alors toutes les causes qui sont directement reliées à ce mode doivent être inculpées.

Règle 2 :

Quand tous les modes de défaillance qui relient directement une cause sont disculpés, alors cette cause est disculpée.

Si on observe seulement le mode de défaillance 1, d'après la règle 1, les causes 1 et 2 doivent être inculpées. Les causes 3, 4 et 5 sont directement disculpées en utilisant la règle 2. Par conséquent, l'AMDEC a diminué le nombre de tests de 5 à 2.

- *L'analyse quantitative*

Suite à une analyse AMDEC, les modes de défaillance sont classés suivant trois critères à savoir :

- La fréquence d'apparition
- La gravité
- La détection

On peut regrouper l'ensemble de ces paramètres (fréquence d'apparition, gravité et détection) en un seul et unique critère, à savoir : la criticité.
On évalue la criticité par le produit :

$$\text{Criticité } = Cr_i = \text{Fréquence d'apparition} * \text{Gravité} * \text{détection}$$

La criticité peut être utilisée comme critère de décision pour générer la séquence de tests qui permet de détecter rapidement le composant responsable de la défaillance. On range les

composants suivant l'ordre décroissant de Cr_i. Le composant possédant la criticité la plus élevée sera testé en premier.

Remarque

Un composant peut avoir une criticité importante alors que sa fréquence d'apparition est faible. En ce moment, il sera plus judicieux de séparer ces trois critères. On change le poids de chaque critère pour accorder un poids plus élevé au critère "fréquence d'apparition" par rapport aux critères "gravité" et "détection". Les composants seront classés en ce moment selon l'ordre décroissant d'un nouveau critère noté K.

$$K = \frac{(\text{fréquence d'apparition} + \text{gravité} + \text{détection})}{3}$$

Sachant que les critères "fréquence d'apparition", "gravité" et "détection" doivent être normalisés suivant leurs poids choisis.

2.6. Les méthodes inductives

La démarche de mise en œuvre est inverse à celle des méthodes de diagnostic déductives. On cherche l'événement possible entraînant l'apparition d'une action indésirable en utilisant une hiérarchie ascendante.

2.6.1. Les arbres de défaillances

La construction d'un arbre de défaillance est la réponse qu'on puisse noter face aux questions suivantes « comment tel événement peut-il se produire ? » ou bien « quelles sont les relations qui peuvent traduire la réalisation d'un tel événement ? »

Cet arbre est souvent présenté de haut en bas. Au sommet, on trouve l'événement qu'on cherche à décrire. Les relations construisant l'arbre de défaillance sont souvent affichées par des liens logiques ET/OU. Partant de la défaillance observée, on suit la logique inductive pour déterminer la cause à l'origine.

Les arbres de défaillances seront présentés dans le chapitre 4 en s'appuyant sur la théorie d'information, les heuristiques et la programmation dynamique.

2.6.2. Le diagramme causes-effet

L'arbre de causes-effet explique un phénomène ou un processus en remontant les causes possibles qui peuvent être à l'origine. Pour une panne donnée, cette représentation construit les causes possibles tout en allant des causes principales vers les petits détails cachés. Ce diagramme forme un outil de diagnostic suffisamment utile pour les personnes n'ayant pas une bonne connaissance dans le domaine en question.

Les méthodes inductives seront utilisées dans le chapitre 5 qui traite les systèmes complexes lorsque plusieurs composants peuvent tomber en panne simultanément.

Chapitre 3 : Approche probabiliste

1. Introduction

La difficulté principale rencontrée lors de la remise en service d'un système hors d'usage est la localisation du composant défaillant. En absence de redondance, le diagramme de structure d'un système constitué de (n) composant est de type série. En effet, pour que le système fonctionne, il faut que tous les composants fonctionnent. La défaillance d'un composant entraîne la défaillance du système. La procédure de tests à effectuer pour localiser le composant défaillant prend fin aussitôt que le composant défaillant est localisé.
Ce chapitre présente une procédure de génération de la séquence de tests à coût minimal en utilisant l'approche probabiliste. On développe des séquences de tests non séquentielles où la procédure de tests est déterminée dès le début de l'inspection.

2. Hypothèses

- Les composants sont stochastiquement indépendants;
- L'état d'un composant n'est connu qu'après le test;
- Les composants sont testés un à la fois;
- Les composants et le système ne peuvent être qu'en état d'opération ou hors d'usage;
- Tous les composants sont accessibles pour effectuer les tests;
- Les tests sont parfaits (on dit qu'un test est parfait si son résultat est certain);
- Le coût (temps) associé à chaque test est connu;
- Soit $p[i|S] = p_i$, la probabilité conditionnelle que le composant i soit défaillant sachant que le système l'est. Cette probabilité est supposée connue dans ce mémoire. En effet, si

S est l'événement "défaillance du système", $p(S)$ est la probabilité que le système soit défaillant et $p[i]$ est la probabilité à priori que le composant i soit responsable de la défaillance. Alors, la probabilité conditionnelle de défaillance p_i est définie comme suit :

$$p_i = \frac{p[i \cap S]}{p[S]}$$

$$p_i = p[S|i] * \frac{p[i]}{\sum_{j=1}^{n} p[S|j] * p[j]}$$

Comme la défaillance du système ne peut être causée que par un seul composant, il en résulte que :

$$\sum p_i = 1 \ \ pour\ i = 1 \dots n$$

3. Contribution au problème de diagnostic

3.1. Séquence optimale de tests

GLUSS et FIRSTMAN [11,12] proposent un modèle qui permet de générer la séquence de tests en utilisant des tests imparfaits (un test est dit imparfait si son résultat est incertain). La séquence générée permet de localiser le composant défaillant à coût minimal. GLUSS et FIRSTMAN [11,12] supposent qu'on connaît le coût associé à chaque composant et la probabilité conditionnelle p_i que le composant (i) soit responsable de la défaillance.

Dans le cas de tests parfaits et sachant qu'un seul composant peut tomber en panne, si après (n-1) tests le composant défaillant n'est pas détecté, il est certain que le composant (n) soit responsable de la défaillance sans avoir besoin de le tester (NACHLAS modifié [20]). Sinon, si le composant défectueux est trouvé parmi les (n-1) premiers composants, il sera de même inutile de tester le composant (n) car la séquence de tests prend fin aussitôt que le composant responsable de la défaillance est détecté. On effectuera au plus (n-1) tests si le système est constitué de (n) composants.

Le coût total moyen de la séquence optimale de tests est défini comme suit:

$$C(S_n) = \ E[C(S_n)] = c_1 \ + \ c_2 * (1 - \ p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) + \ c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \quad (3.1)$$

Sachant que c_i représente le coût encouru pour tester l'ième élément de la séquence de tests.

Théorème 3.1

Selon CANFIELD et NACHLAS [20], la séquence de tests S_n^* qui minimise le coût total moyen est celle obtenue en ordonnant les composants suivant le rapport p_i/c_i décroissant:

$$\frac{p_i}{c_i} > \frac{p_{i+1}}{c_{i+1}} \ pour \ i \in [1..n-2] \quad (3.2)$$

Preuve

Soit :

- $S_n = [(1), (2) \ldots (j-1), (j), (j+1) \ldots (n-1)]$ la séquence qui teste le composant (j) avant le composant (j+1)
- $\bar{S_n} = [(1), (2) \ldots (j-1), (j+1), (j) \ldots (n-1)]$ la séquence qui teste le composant (j+1) avant le composant (j)

On définit le coût total moyen associé aux deux séquences S_n et $\bar{S_n}$:

$$C(S_n) = \ E[C(S_n)] = c_1 \ + \ c_2 * (1 - \ p_1) + \cdots + c_j * \left(1 - \sum_{i=1}^{j-1} p_i\right) + c_{j+1} * \left(1 - \sum_{i=1}^{j} p_i\right) + \cdots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \quad (3.3)$$

$$C(\bar{S}_n) = E\left[C(\bar{S}_n)\right] = c_1 + c_2 * (1 - p_1) + \cdots + c_{j+1} * \left(1 - \sum_{i=1}^{j-1} p_i\right) + c_j * \left(1 - (\sum_{i=1}^{j-1} p_i) - p_{j+1}\right) + \cdots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \quad (3.4)$$

ainsi,

$$C(S_n) - C(\bar{S}_n) = p_{j+1} * c_j - p_j * c_{j+1} \quad (3.5)$$

La permutation entre les composants (j) et (j+1) sera avantageuse seulement si $C(S_n) > C(\bar{S}_n)$. Ceci montre que la séquence optimale de tests est celle qui vérifie :

$\frac{p_i}{c_i} > \frac{p_{i+1}}{c_{i+1}}$ pour $i \in [1..n-2]$

3.2. Cas de la décomposition en modules et en composants

Si le système comporte (m) modules opérant suivant une structure série, il sera plus commode si on admet la notion de modularité. Ainsi, si on connaît c_I^m (le coût de tests du module (I)) et p_I^m (la probabilité conditionnelle que le module (I) soit défaillant), on applique ainsi la procédure de tests du théorème 3.1 pour localiser le module défaillant. Ensuite on teste à nouveau ces modules pour déterminer le composant responsable de la défaillance.

3.3. Modèle p/c avec coût de désassemblage

Dans le cas où un coût de désassemblage est encouru pour tester un module, BHANDARI [3] suggère de tester tous les composants d'un même module avant de passer aux autres. Ce modèle reprend les mêmes hypothèses du modèle p/c en rajoutant les équations suivantes :

$$c_{(I)} = c_{d(I)} + E\left[C\left(S_{m(I)}\right)\right] \quad (3.6)$$

$$p_{(I)} = \sum_{n=1}^{N(I)} p_{m(I)} \tag{3.7}$$

$c_{d(I)}$ est le coût de désassemblage du module (I), $m(I)$ représente les composants du module (I) et $N(I)$ représente le nombre de composants du module (I)

3.4. Modèle p/c modifié

Plus tard, dans un second travail, CANFIELD et NACHLAS [4] ont rapporté un contre-exemple simple basé sur le Tableau 1:

Tableau 1: Contre exemple

Composant	$P_{(i)}$	$C_{(N)}$	$P_{(i)}/C_{(i)}$
1	0.65	5.0	0.13
2	0.25	2.0	0.125
3	0.1	1.0	0.100

Comme GIRAUD [10] l'a expliqué, si on utilise la procédure de tests du théorème 3.1, la séquence optimale de tests sans tester le dernier composant est [(1),(2),(3)] avec un coût total moyen égal à 5.7. Mais, la séquence optimale est [(2),(3),(1)] avec un coût total moyen égal à 2.75. L'erreur résulte du fait que la procédure de tests du théorème 3.1 ne tient pas en compte du non-test du dernier composant de la séquence. De ce fait, CANFIELD et NACHLAS [4] ont rajouté la condition $c_{n-1} < c_n$ pour que la séquence de tests obtenue à partir du ratio p/c minimise le coût total moyen.

Théorème 3.2

L'algorithme qui offre la séquence optimale de tests S_n^* est celui qui vérifie l'équation (3.9) où les composants sont ordonnés suivant le rapport p/c décroissant. Comme indique l'équation (3.8), on ne teste pas le composant (n) tant que son coût de test est supérieur à

celui du composant (n-1) :

$$\begin{cases} c_{n-1} < c_n & (3.8) \\ \frac{p_i}{c_i} > \frac{p_{i+1}}{c_{i+1}} \; pour \; i \in [1..n-2] & (3.9) \end{cases}$$

Il ne faut pas oublier que le coût de test du composant (n) n'apparaît pas dans l'expression du coût total moyen:

$$\begin{aligned} C(S_n) = \; E[C(S_n)] = c_1 \; + \; c_2 * (1 - \; p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) \\ + \; c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \end{aligned} \quad (3.10)$$

Preuve de l'équation (3.8)

Soit la séquence S'_n = [(1),(2)…(j-1),(j),(j+1)…(n),(n-1)] formée en changeant la position des deux composants adjacents (n-1) et (n) de la séquence optimale de tests $S_n{}^*$. Il faudra que la nouvelle séquence ait un coût moyen supérieur à celle de la séquence optimale. Alors par hypothèse, $E[C(S_n{}^*)] < E[C(S'_n)]$.

D'après l'équation (3.4) :

$$\begin{aligned} C(S_n{}^*) = \; E[C(S_n{}^*)] = c_1 \; + \; c_2 * (1 - \; p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) \\ + \; c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) \ldots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \end{aligned}$$

et :

$$\begin{aligned} C(S'_n) = \; E[C(S'_n)] = c_1 \; + \; c_2 * (1 - \; p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) \\ + \; c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) \ldots + c_n * \left(1 - \sum_{i=1}^{n-2} p_i\right) \end{aligned} \quad (3.11)$$

ainsi,

$$E[C(S_n{}^*)] - E[C(S'_n)] = (c_{n-1} - c_n) * (1 - p_1 - \cdots - p_{n-2}) \quad (3.12)$$

Pour que $S_n{}^*$ soit optimale, il faut que $(c_{n-1} - c_n) * (1 - p_1 - \cdots - p_{n-2}) < 0$. La somme $(1 - p_1 - \cdots - p_{n-2})$ est positive, il est clair ainsi que $(c_{n-1} - c_n) < 0$ (condition vérifiée).

3.5. Contre exemple du modèle p/c modifié

Partant des travaux faits par CANFIELD et NACHLAS [4], un contre-exemple est présenté dans le mémoire de GIRAUD [10] pour étudier la validité de la simplification qui consiste à ne plus tester le dernier composant. Soit le Tableau 2 suivant :

Tableau 2 : Contre exemple du modèle p/c modifié

I	Pi	C_i	P_i/C_i	E(C(S))	I	P_i	C_i	P_i/C_i	E(C(S))
1	0.2740	2	0.1370	2.0000	**1**	0.2740	2	0.1370	2.0000
2	0.0702	1	0.0702	2.7260	**2**	0.0702	1	0.0702	2.7260
3	0.1897	3	0.0632	4.6934	**3**	0.1897	3	0.0632	4.6934
4	0.0594	1	0.0594	5.1595	**4**	0.0594	1	0.0594	5.1595
5	**0.1371**	**5**	**0.0274**	**7.1930**	**6**	0.0526	2	0.0263	5.9729
6	0.0526	2	0.0263	7.7322	**7**	0.0938	4	0.0235	7.3893
7	0.0938	4	0.0235	8.6002	**8**	0.0574	3	0.0191	8.1702
8	0.0574	3	0.0191	8.9698	**9**	0.0600	4	0.0150	8.9818
9	0.0600	4	0.0150	9.2330	**10**	0.0058	4	0.0015	9.5534
10	0.0058	4	0.0015	-	**5**	**0.1371**	**5**	**0.0274**	-
Somme	1	29	-	**9.2330**	**Somme**	1	29	-	**9.5534**

Bien que la partie gauche du tableau 2 ne respecte pas la condition (3.8), elle présente un coût total moyen inférieur à celui de la partie droite.

Dans son travail [10], GIRAUD indique les erreurs faites dans le dernier modèle. Il explique la généralité des conditions (3.8) et (3.9) et propose le choix des inégalités au sens

large dans les équations qui viennent. De plus, il met l'accent sur le fait que le test du dernier composant de la séquence n'est pas requis. GIRAUD précise aussi que la séquence obtenue en déplaçant l'un des composants en dernière position doit être directement issue de la séquence optimale.

Théorème 3.3

La séquence optimale de tests est celle qui vérifie les conditions suivantes :

$$\begin{cases} c_{n-1} \leq c_n & (3.13) \\ \frac{p_i}{c_i} > \frac{p_{i+1}}{c_{i+1}} \; pour \; i \in [1..n-2] & (3.14) \end{cases}$$

Remarque

L'équation (3.8) rajoutée par CANFIELD et NACHLAS [4] et corrigée par GIRAUD [10] exige que le coût encouru pour tester le composant (n) doive être supérieur ou égal à celui requis pour tester le composant (n-1). Le Tableau 3 sera présenté pour discuter la validité du théorème 3.3:

Tableau 3 : Contre exemple du théorème 3

Composant	$P_{(i)}$	$C_{(N)}$	$P_{(i)}/C_{(i)}$
1	0.65	5.0	0.13
2	0.14	1.12	0.125
3	0.21	2.1	0.100

On constate que la séquence [(1),(2),(3)] suit les conditions d'optimalité du théorème 3.3 avec un coût total moyen égal à 5.39, mais, la séquence optimale est [(3),(2),(1)] avec un coût total moyen égal à 2.92.

Il s'agit d'un problème NP complet. L'approche utilisée est fondamentalement basée sur le modèle du sac à dos. Une séquence S_1 est préférable à une séquence S_2 si $C(S_1) < C(S_2)$. Il

est bien entendu que si le nombre de composants à tester est petit, on procédera par énumération.

4. Séquence dominante

Une séquence de tests est dite dominante si son coût total moyen est inférieur à celui de la deuxième séquence. Dans sa thèse, GAO [8] définit la séquence de tests S_k = [(1),(2)…(k-1),(k+1)…(n-1),(n)]. Cette séquence est obtenue en supprimant le composant (k) (k ϵ [1…n-2]) de la séquence formée des composants ordonnés suivant l'ordre décroissant de p/c. Le coût total moyen encouru pour séquence de tests S_k est donné par :

$$C(S_k) = E[C(S_k)] = c_1 + c_2 * (1 - p_1) + \cdots + c_{k-1} * \left(1 - \sum_{i=1}^{k-2} p_i\right) \tag{3.19}$$
$$+ c_{k+1} * (1 - \sum_{\substack{i=1 \\ i \neq k}}^{k} p_i) + \cdots + c_{n-1} * (1 - \sum_{\substack{i=1 \\ i \neq k}}^{n-2} p_i) + c_n * (1 - \sum_{\substack{i=1 \\ i \neq k}}^{n-1} p_i)$$

Le coût encouru pour tester le composant (k) n'apparaît pas dans l'équation (3.19) et ceci est du qu'il est retiré de la séquence.

En rajoutant et en retranchant le terme $p_k * \sum_{i=k+1}^{n} c_i$, l'expression (3.19) devienne :

$$C(S_k) = E[C(S_k)] = c_1 + c_2 * (1 - p_1) + \ldots + c_{k-1} * \left(1 - \sum_{i=1}^{k-2} p_i\right) \tag{3.21}$$
$$+ c_{k+1} * (1 - \sum_{i=1}^{k} p_i + p_k) + \cdots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i + p_k\right) + c_n * \left(1 - \sum_{i=1}^{n-1} p_i + p_k\right)$$

- Si $c_n > c_k$, GAO [8] définit la séquence de tests S_k^n en remplaçant le composant (n) par le composant (k) dans la séquence S_k. Ainsi, S_k^n = [(1),(2)…(k-1),(k+1)…(n-1), (k)].

Le coût total moyen encouru pour la séquence S_k^n est donné par:

$$C(S_k^n) = E[C(S_k^n)] = c_1 + c_2 * (1 - p_1) + \cdots + c_{k-1} * \left(1 - \sum_{i=1}^{k-2} p_i\right) \tag{3.22}$$
$$+c_{k+1} * \left(1 - \sum_{i=1}^{k} p_i + p_k\right) + \cdots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i + p_k\right) + c_k * \left(1 - \sum_{i=1}^{n-1} p_i + p_k\right)$$

à partir des équations (3.21) et (3.22) on aura :

$$C(S_k^n) - C(S_k) = c_k * \left(1 - \sum_{i=1}^{n-1} p_i + p_k\right) - c_n * \left(1 - \sum_{i=1}^{n-1} p_i + p_k\right)$$
$$C(S_k^n) - C(S_k) = (c_k - c_n) * \left(1 - \sum_{i=1}^{n-1} p_i + p_k\right) \tag{3.23}$$
$$C(S_k^n) - C(S_k) = (c_k - c_n) * (p_n + p_k)$$

Sachant que $c_n > c_k$, et que $(p_n + p_k)$ est positif, alors $C(S_k^n) < C(S_k)$ et la séquence S_k^n est meilleure que S_k . On dit que la séquence S_k^n domine S_k .

Par la suite, on revient à la séquence optimale de tests formée des (n) composants ordonnés suivant le rapport p/c, S_n =[(1),(2)…(k-1),(k),(k+1)… (n-1)].
D'après (3.3), le coût total moyen associé à la séquence S_n est :

$$C(S_n) = E[C(S_n)] = c_1 + c_2 * (1 - p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right)$$
$$+ \cdots + c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right)$$

- Si $c_n < c_{n-1}$, on définit d'une manière similaire, S_{n-1} = [(1),(2)…(k-1),(k), (k+1)…(n-2),(n)]. Le coût total moyen encouru pour la séquence S_{n-1} est donné par:

$$C(S_{n-1}) = E[C(S_{n-1})] = c_1 + c_2 * (1 - p_1) \quad (3.25)$$
$$+ \cdots + c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) + c_n * \left(1 - \sum_{i=1}^{n-2} p_i\right)$$

en utilisant les équations (3.3) et (3.25):

$$C(S_{n-1}) - C(S_n) = c_n * \left(1 - \sum_{i=1}^{n-2} p_i\right) - c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right) \quad (3.26)$$

alors,

$$C(S_{n-1}) - C(S_n) = (c_n - c_{n-1}) * (p_n + p_{n-1}) \quad (3.27)$$

Sachant que $c_n < c_{n-1}$ et que $(p_n + p_{n-1})$ est positif, on conclu que $C(S_{n-1}) < C(S_n)$. Alors, la séquence S_{n-1} est meilleure que S_n. On dit que S_{n-1}domine S_n.

5. Le critère de la séquence dominante de tests

- Si $c_n > c_{n-1}$ et $c_n > c_k$ *(k < n)*

Lemme

Si $c_n > c_k$ alors, S_n domine S_k^n

On a vu précédemment que si $c_n > c_k$, alors la séquence S_k^n domine S_k . Il en résulte que si $c_n > c_{n-1}$, alors S_n domine S_{n-1}. En plus, à partir des équations (3.4) et (3.19) on peut calculer :

$$C(S_n) - C(S_k^n) = C_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) - [C_k * \left(1 - \sum_{i=1}^{k-1} p_i + p_k\right) + p_k * \sum_{i=k+1}^{n-1} c_i\ ]$$

$$C(S_n) - C(S_k^n) = C_k * \left(\sum_{i=k+1}^{n-1} p_i \right) - p_k * \sum_{i=k+1}^{n-1} c_i \tag{3.28}$$

Preuve

On range les composants dans l'ordre des $\frac{p_i}{c_i}$ décroissant, alors :

$$\frac{p_k}{c_k} \geq \frac{p_{k+1}}{c_{k+1}} \geq \cdots \geq \frac{p_{n-1}}{c_{n-1}} \text{ avec k = 1,2…n} \tag{3.29}$$

on peut écrire l'équation précédente de la façon suivante :

$$\begin{cases} p_k * c_{k+1} \geq c_k * p_{k+1} \\ p_k * c_{k+2} \geq c_k * p_{k+2} \\ p_k * c_{n-1} \geq c_k * p_{n-1} \end{cases} \tag{3.30}$$

sommons ces équations:

$$p_k * \sum_{i=k+1}^{n-1} c_i \geq c_k * \left(\sum_{i=k+1}^{n-1} p_i \right) \tag{3.31}$$

alors S_n domine S_k^n

- Si $c_n < c_{n-1}$ et $c_n > c_k$

Lemme

Si $c_n > c_k$ alors, S_k^n domine S_k ;

Si $c_n < c_{n-1}$ alors, S_{n-1}domine S_n

En effet, si $c_n > c_k$ et $c_n < c_{n-1}$, alors, les deux séquences dominantes seront S_k^n et S_{n-1}. Sachant que S_n domine S_k^n , on peut conclure que S_{n-1} domine S_n qui, domine à son tour la séquence S_k^n qui domine la séquence S_k.

Plusieurs comparaisons ont été faites. En effet, si $c_n > c_k$ et $c_n < c_{n-1}$, alors la séquence S_n sera la meilleure séquence. Pour les autres cas, on ne peut vraiment pas tirer de conclusions en ce qui concerne l'optimalité.

Théorème 3.4

Soit l'ensemble Ω = [S_k / $c_k > c_n$] pour tout k = 1, 2, …, n-1. Selon GAO [8], la meilleure séquence de tests peut-être S_n ou incluse dans l'ensemble Ω. Si c_n = Max $_{i \in [1....n]}\{ c_i \}$, S_n est la meilleure séquence.

La Figure 3 résume les modèles déjà présentés qui permettent de générer la séquence de tests à effectuer.

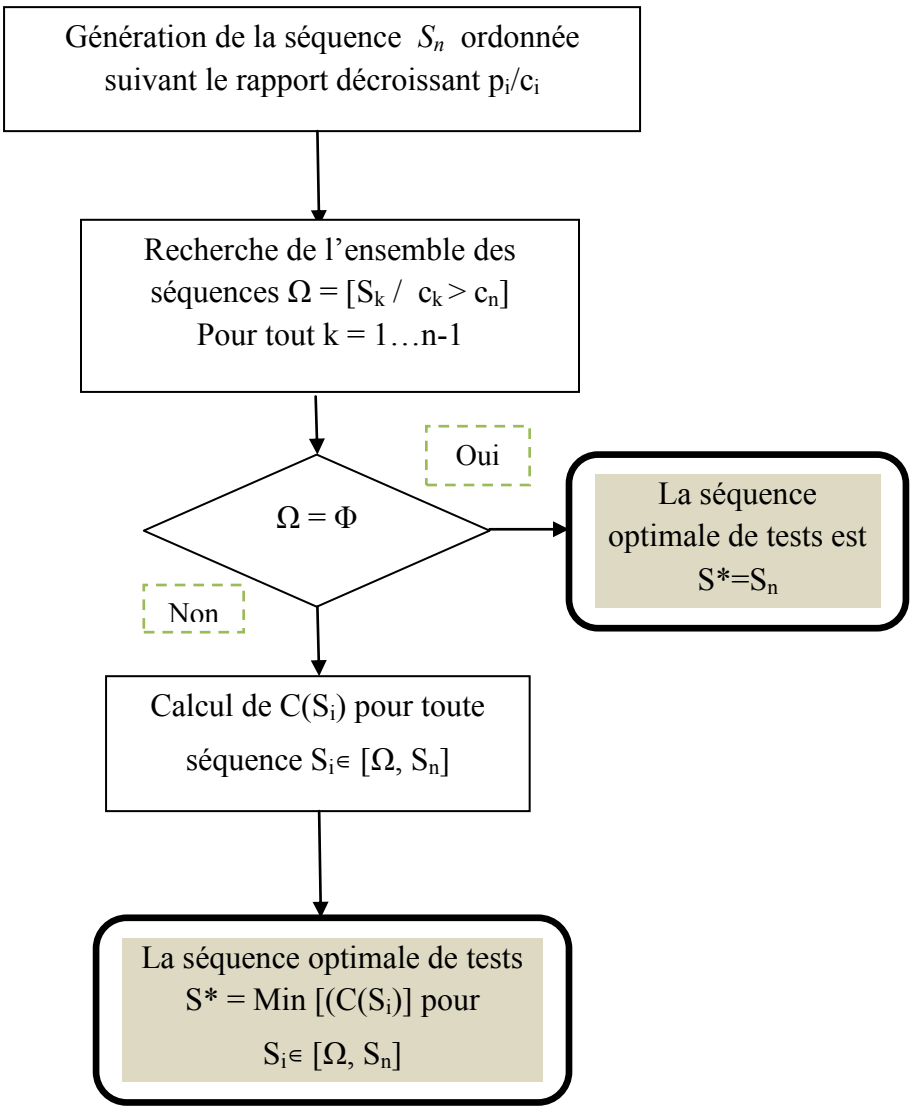

Figure 3. Procédure de génération de la séquence de tests connaissant le couple (p_i, c_i) pour chaque composant

6. Cas particuliers

Cas 1 : $p_j/c_j = p_{j+1}/c_{j+1}$ pour tout j = 1,2…n

Si $p_j/c_j = p_{j+1}/c_{j+1}$ pour tout j = 1,2…n, NACHLAS et al [4] confirme que la permutation des deux composants (j) et (j+1) avec $j \in [1, n-1]$ ne modifie pas le coût total moyen.

Démonstration

Déterminons la différence de coûts entre les deux séquences S_n et $\widetilde{S_n}$ = [(1),(2)…(j-1) ,(j+1),(j)…(n-1)].

$$C(\widetilde{S_n}) = E[C(\widetilde{S_n})] = c_1 + c_2 * (1 - p_1) + \cdots + c_{j+1} * \left(1 - \sum_{i=1}^{j-1} p_i\right) \quad (3.32)$$

$$+ c_j * (1 - \sum_{i=1}^{j-1} p_i - p_{j+1}) + \cdots + c_{n-1} * (1 - \sum_{i=1}^{n-2} p_i)$$

d'après (3.3) :

$$C(S_n) = E[C(S_n)] = c_1 + c_2 * (1 - p_1) + \cdots + c_k * \left(1 - \sum_{i=1}^{k-1} p_i\right)$$

$$+ c_{n-2} * \left(1 - \sum_{i=1}^{n-3} p_i\right) + \ldots + c_{n-1} * \left(1 - \sum_{i=1}^{n-2} p_i\right)$$

alors :

$$C(S_n) - C(\widetilde{S_n}) = p_j * c_{j+1} - p_{j+1} * c_j = 0 \quad (3.33)$$

Cas 2 : $p_k = p_{k+1}$ ou $c_k = c_{k+1}$ pour tout k = 1,2…n

Selon GAO [8], la différence de coûts entre les deux séquences de tests S_k et S_{k+1} est :

$$C(S_k) - C(S_{k+1}) = (c_{k+1} - c_k) * \left(\sum_{i=k}^{n} p_i\right) + (p_k - p_{k+1}) * \left(\sum_{i=k+1}^{n-1} c_i\right) \quad (3.34)$$

Dans le cas où $c_{k+1} = c_k$, le signe de $C(S_k) - C(S_{k+1})$ dépend du signe de $(p_k - p_{k+1})$.

on sait que :

$$\frac{p_k}{c_k} \geq \frac{p_{k+1}}{c_{k+1}} \tag{3.35}$$

alors,

$$p_k > p_{k+1} \tag{3.36}$$

ainsi :

$$C(S_k) > C(S_{k+1}) \tag{3.37}$$

Le même raisonnement est utilisé si $p_k = p_{k+1}$

Ainsi**,** pour chaque séquence de tests S_k qui satisfait les conditions d'optimalité, si $p_k = p_{k+1}$ où $c_k = c_{k+1}$, $C(S_k) > C(S_{k+1})$, donc la séquence S_{k+1} domine S_k

Conclusion

Dans ce chapitre, la connaissance du couple (p_i , c_i) pour chaque composant (i) où, p_i est la probabilité conditionnelle que le composant i soit défaillant sachant que le système l'est et c_i est le coût encouru pour tester le composant (i), a permis de générer la séquence de (n-1) tests qui minimise le coût total moyen pour localiser le composant responsable de la défaillance. La génération de cette séquence est basée sur le ratio $\frac{p_i}{c_i}$ et sur les relations entre les différends p_i d'une part et les c_i d'autre part. La séquence de tests prend fin aussitôt que le composant responsable de la défaillance est localisé.

Les tests sont parfaits. La séquence demeure inchangée. Elle n'est pas modifiée à l'issue d'un test.

Chapitre 4 : Approche basée sur la théorie d'information et la programmation dynamique

1. Introduction

Le chapitre précédent a permis de générer la séquence de tests à effectuer en se basant sur le ratio p_i/c_i ; i =1,2…n. Dans ce chapitre, on se propose d'utiliser la théorie d'information et la programmation dynamique. La théorie d'information est apparue avec VASHENEY et HARTMAN [26]. Elle est basée sur l'information qu'on peut gagner après chaque test. Le test ne détecte pas forcément la panne, mais peut nous guider pour le bon choix des prochains tests. Alors, la séquence n'est pas fixée au début de l'inspection. Elle est générée au fur et à mesure que les tests sont effectués.

Au début des années 1990, PATTIPATI et al. [21] ont repris les mêmes bases utilisées par VASHENEY et HARTMAN [27]. Ils ont proposé un algorithme qui permet de détecter tous les défauts individuels possibles en utilisant un dictionnaire de tests. Un dictionnaire de tests est un tableau qui regroupe les états de défaillance, leur probabilité d'occurrence, les tests disponibles et le coût associé à chacun d'eux. Ce tableau indique si le test t_j est capable de détecter si le système est dans l'état S_i ou non (voir tableau 4). GAREY et GRAHAM [9] et GIRAUD [10] considèrent C_m comme un coût habituel de l'application du test qui est multiplié par le nombre de fois que le test est répété. Le test t_1 peut détecter par exemple les états de défaillance S_4, S_5 et S_6 et nous coûtera $1^{\$}$ pour l'effectuer.

Remarque

Dans ce mémoire, il serait plus judicieux d'interpréter les S_i comme des causes responsables de la défaillance du système. On cherche la séquence de tests permettant de tomber rapidement sur la cause de défaillance. On utilise des tests binaires. Pour un test t_j, si le dictionnaire de tests comprend des zéros partout dans la colonne spécifiée, alors le test t_j ne peut détecter aucune cause. De même, si le dictionnaire de tests indique des 1 partout dans la colonne spécifiée au test t_j, alors celui-ci peut détecter toutes les causes responsables de la défaillance du système. Pour une matrice diagonale, toutes les causes sont détectées par un test unique.

Tableau 4 : Dictionnaire de tests

Causes de défaillance / *Tests*	t_1	t_2	t_3	t_4	t_5	*Probabilités de défaillance*
S_1	0	0	0	0	0	**0.07**
S_2	0	1	0	0	1	**0.01**
S_3	0	0	1	1	0	**0.02**
S_4	1	0	0	1	1	**0.1**
S_5	1	1	0	0	0	**0.05**
S_6	1	1	1	1	0	**0.12**
Le coût du test	1	3	5	1	2	**$\sum = 1$**

Plus tard, PATTIPATI et al. [23] ont poursuivi leurs recherches avec l'arrivée du logiciel START. Comme GIRAUD [10] l'a expliqué, ce logiciel utilise des algorithmes basés sur la théorie d'information, la recherche heuristique et la théorie des graphes. START traite des différentes facettes du problème de génération de la séquence de tests et du problème de l'analyse de testabilité. L'efficacité du logiciel START apparaît dans plusieurs domaines du diagnostic et l'approche utilisée peut être adaptée à divers problèmes tels que le diagnostic médical, le traitement de données et la reconnaissance de formes.

2. Description du système à étudier

- Le système étudié est constitué de m composants;
- $S= [S_1, S_2 \ldots, S_m]^T$ désigne l'ensemble de causes responsables de la défaillance du système. À un instant donné, la défaillance du système est provoquée par une seule cause;
- $p(S_i)_{i=1..m}$ désigne la probabilité que la cause S_i soit responsable de la défaillance du système ;
- $T= [t_1, t_2 \ldots, t_n]^T$ désigne l'ensemble de tests disponibles;
- $C= [c_1, c_2 \ldots, c_n]^T$ désigne l'ensemble de coûts encourus en appliquant les tests de T;
- $D= [d_{ij}]_{m*n}$ désigne la matrice de diagnostic sachant que :

$$d_{ij(\text{ pour i=1...m, j=1...n})} = \begin{cases} 1 & \text{si la cause } S_i \text{ peut} - \text{être détecté par le test } t_j \\ 0 & \text{Sinon} \end{cases}$$

Un test t_j partage un ensemble de causes de défaillance x en deux sous-ensembles x_{jf} et x_{jp}. Ces ensembles sont définis comme suit :

$$x_{jf} = \{S_i | \, d_{ij} = 1, \, S_i \in x\} \tag{4.1}$$

$$x_{jp} = \{S_i | \, d_{ij} = 0, \, S_i \in x\} \tag{4.2}$$

Les sous-ensembles x_{jf} et x_{jp} correspondent respectivement à l'échec et aux sucées du test t_j. L'ensemble x_{jf} regroupe les causes de défaillance contenues dans x et qui peuvent être détectées par le test t_j alors que l'ensemble x_{jp} est formé des causes de défaillances contenues dans x et qui ne peuvent pas être détectées par le même test. Il faut mentionner que $x = x_{jf} \; U \; x_{jp}$ et que $x_{jf} \cap x_{jp} = \Phi$. Les lettres f et p correspondent à l'échec (*fail*) et

aux sucées (*pass*) du test t_j. La séquence de tests peut être schématisée par un arbre de décision binaire. Le nœud d'origine situé en haut de l'arbre contient toutes les causes possibles provoquant la défaillance système. Les nœuds en bas correspondent aux causes individuelles de défaillances. Un arbre de décision binaire est formé comme illustre la Figure 4.

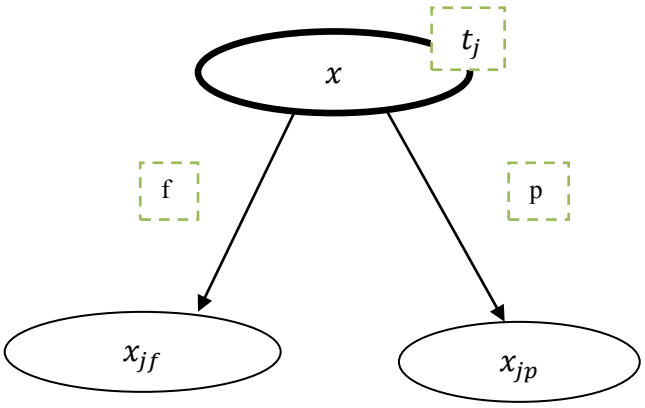

Figure 4. Procédure de génération d'un arbre de décision

Les probabilités $p(x_{jf})$ et $p(x_{jp})$ sont données par:

$$p(x_{jf}) = \left[\sum_{Si \in x} d_{ij} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] \tag{4.3}$$

$$p(x_{jp}) = 1 - p(x_{jf}) \tag{4.4}$$

La probabilité conditionnelle de défaillance de chaque nœud est la somme des probabilités conditionnelles de défaillance des causes incluses dedans. Particulièrement, pour le nœud initial lorsque le système est défaillant:

$$\sum_{i=1}^{m} p(S_i) = 1 \tag{4.5}$$

3. Exemple 1

Prenons l'exemple tiré de la thèse de GAO [8]. On dispose de cinq causes de défaillance S = [S_1, S_2…, S_5]. Cinq tests T= [t_1, t_2…, t_5] sont disponibles. Le Tableau 5 regroupe les probabilités conditionnelles des causes de défaillance, les coûts des tests et leurs rapports avec chaque cause:

Tableau 5 : Dictionnaire de tests de l'exemple 1

Causes de défaillance / *Tests*	t_1	t_2	t_3	t_4	t_5	*Probabilités de défaillance*
S_1	0	1	0	0	1	**0.3**
S_2	0	0	1	1	0	**0.25**
S_3	1	0	0	1	1	**0.2**
S_4	1	1	0	0	0	**0.1**
S_5	1	1	1	1	0	**0.15**
Le coût du test	6.5	5	4	3	6	**$\sum = 1$**

L'application du premier test t_1 donne naissance aux deux nœuds [S_3, S_4, S_5] et [S_1, S_2] correspondant à l'échec et aux sucées du test. On cherche ensuite les tests appropriés pour séparer les deux nœuds formés. On applique le test t_3 pour séparer les causes [S_1, S_2] et le test t_2 pour les causes [S_3, S_4, S_5]. Si le résultat de test t_3 est négatif, la cause S_2 est alors inculpée, sinon si le résultat est positif, alors la cause S_2 est disculpée. On fait de même pour le nœud contenant les causes [S_3, S_4, S_5]. On continu le travail jusqu'a atteindre les nœuds terminaux. On présente les résultats dans Figure 5. On note R=0 si le résultat du test est positif et R=1 pour un résultat négatif.

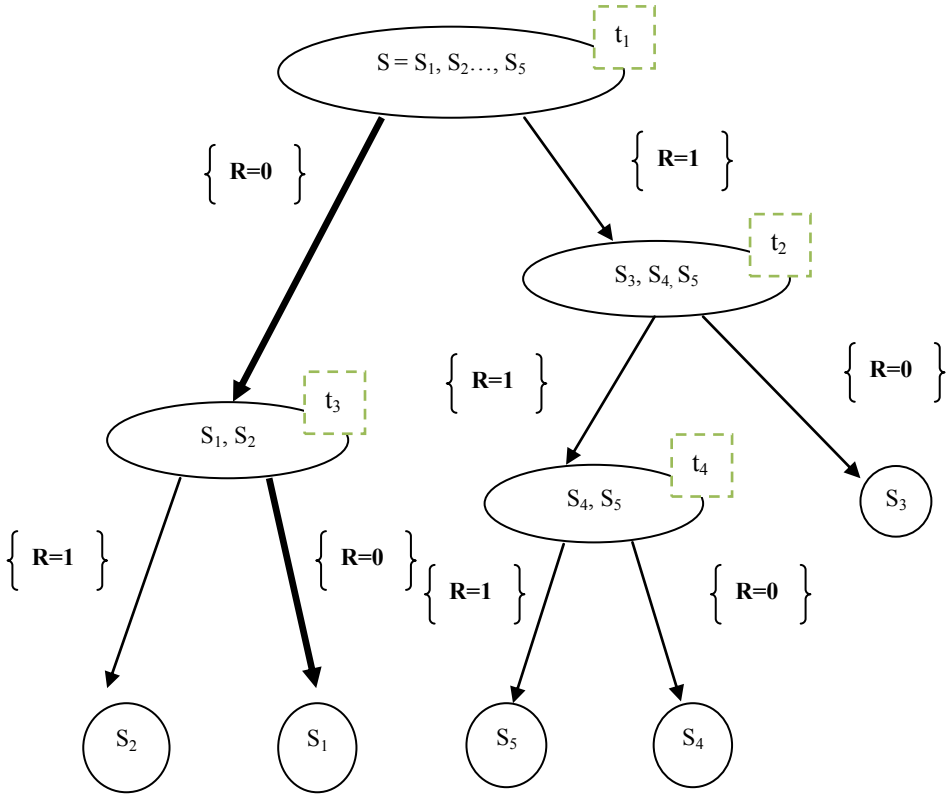

Figure 5. Génération de l'arbre de décision.

Le coût total prévu

Soit TC le coût total prévu. Le coût encouru pour identifier la cause de défaillance S_i est le produit de la probabilité conditionnelle d'occurrence de cette cause par la somme des coûts des tests utilisés pour l'atteindre:

$$c_i = p(S_i) * \sum_{j=1}^{n} a_{ij}.c_j \tag{4.6}$$

sachant que:

$$a_{ij} = \begin{cases} 1 & \text{si le teste } t_j \text{ est utilisé pour identifier la cause } S_i \\ 0 & \text{Sinon} \end{cases}$$

alors, le coût total prévu encouru pour identifier toutes les causes S_i du système est :

$$TC = \sum_{i=1}^{m} c_i \tag{4.7}$$

$$TC = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}.c_j \ .p(S_i) \tag{4.8}$$

calculons le coût total prévu pour l'arbre de décision de la Figure 5:

$cp_1 = p_1 * (c_1 + c_3)$ *(*cp_1 est le coût de tests encouru pour identifier la cause de défaillance S_1)

$cp_2 = p_2 * (c_1 + c_3)$

$cp_3 = p_3 * (c_1 + c_2)$

$cp_4 = p_4 * (c_1 + c_2 + c_4)$

$cp_5 = p_5 * (c_1 + c_2 + c_4)$

Le coût total prévu est la somme des coûts encourus par tous les chemins formant l'arbre de décision :

$CT = cp_1 + cp_2 + cp_3 + cp_4 + cp_5$

$CT = p_1 * (c_1 + c_3) + p_2 * (c_1 + c_3) + p_3 * (c_1 + c_2) + p_4 * (c_1 + c_2 + c_4)$

$+ p_5 * (c_1 + c_2 + c_4)$

$CT = (p_1 + p_2) * (c_1 + c_3) + p_2 * (c_1 + c_2) + (p_4 + p_5) * (c_1 + c_2 + c_4)$

$CT = 11.57$

Remarque :

Le problème réside dans le choix des tests. Pourquoi au début de l'exemple 1 on a choisi le test t_1? Si on change le choix des tests, cela pourrait diminuer le coût d'inspections? Ainsi, le coût total prévu dépend de la façon avec laquelle l'arbre est construit. De ce fait, on s'intéresse ensuite à déterminer la procédure qui permet de détecter le test à effectuer en minimisant le coût prévu d'inspection.

3.1. Les algorithmes proposés

3.2. Algorithme de séparation maximale (MSA)

Selon GAO [8], le gain d'information apporté par l'application du test t_j au nœud x est évalué par la fonction $d\left(x, t_j\right)$. Pour chaque nœud x de l'arbre de décision, le test choisi est celui qui maximise $d\left(x, t_j\right)$ tel que:

$$d\left(x, t_j\right) = p\left(x_{jf}\right) * p\left(x_{jp}\right) \quad \text{pour tout } j \in [1 \ldots n] \tag{4.9}$$

si on substitue l'équation (4.4) dans (4.9), on aura :

$$d\left(x, t_j\right) = p\left(x_{jf}\right) - p\left(x_{jf}\right)^2 \tag{4.10}$$

ou bien :

$$d\left(x, t_j\right) = p\left(x_{jp}\right) - p\left(x_{jp}\right)^2 \tag{4.11}$$

On définit les dérivées partielles de $d\left(x, t_j\right)$ par rapport à $p\left(x_{jf}\right)$ et $p\left(x_{jp}\right)$:

$$\frac{\partial(d(x, t_j))}{\partial x(p(x_{jf}))} = 1 - 2 * p\left(x_{jf}\right) \tag{4.12}$$

$$\frac{\partial(d(x,t_j))}{\partial x(p(x_{jp}))} = 1 - 2 * p(x_{jp}) \tag{4.13}$$

$d(x,t_j)$ est maximale si :

$p(x_{ip}) = p(x_{if}) = 0.5$

3.3. Algorithme de séparation maximale et de coût minimal (MSMCA)

L'algorithme précédant considère seulement les probabilités de défaillance et ne tient pas en compte des coûts de tests. Cependant, le modèle présenté peut être utile si on ne connaît pas les coûts d'inspection ou si les tests possèdent des coûts égaux. Sinon, ces coûts doivent être introduits dans l'équation (4.4). Ainsi, pour un nœud *x*, GAO [8] choisit un test t_j s'il maximise l'information heuristique $d_c(x,t_j)$ tel que:

$$d_c(x,t_j) = [p(x_{jf}) * p(x_{jp})]/c_j \; pour \; tout \; j \in [1 \ldots n] \tag{4.14}$$

en substituant l'équation (4.4) dans l'équation (4.14), on aura :

$$d_c(x,t_j) = [p(x_{jf}) - p(x_{jf})^2]/c_j \tag{4.15}$$

les dérivées partielles de $d_c(x,t_j)$ par rapport à $p(x_{jf})$ et $p(x_{jp})$:

$$\frac{\partial(d_c(x,t_j))}{\partial x(p(x_{jf}))} = \frac{1 - 2 * p(x_{jf})}{c_j} \tag{4.16}$$

$$\frac{\partial(d_c(x,t_j))}{\partial x(p(x_{jp}))} = \frac{1 - 2 * p(x_{jp})}{c_j} \tag{4.17}$$

la dérivée partielle de $d_c(x, t_j)$ par rapport à c_j est :

$$\frac{\partial(d_c(x,t_j))}{\partial c_j} = \frac{(p(x_{jf})^2 - p(x_{jf}))}{{c_j}^2} \tag{4.18}$$

L'équation (4.16) s'annule si $p(x_{jf})$ et $p(x_{jp})$ sont égaux à 0.5 alors que l'équation (4.18) s'annule si $p(x_{jf}) = 0$ ou $p(x_{jf}) = 1$. L'équation qui vérifie les deux conditions est celle qui satisfait la séparation maximale avec un coût minimum. Dans la Figure 6, on va résumer les deux algorithmes présentés précédemment. Ce résumé se base sur les étapes de GAO [8] en rajoutant quelques modifications reflétant l'aspect algorithmique de la démarche. Une structure itérative, compréhensible et simple à utiliser est mise en œuvre. Des conditions initiales et plusieurs paramètres sont posés.

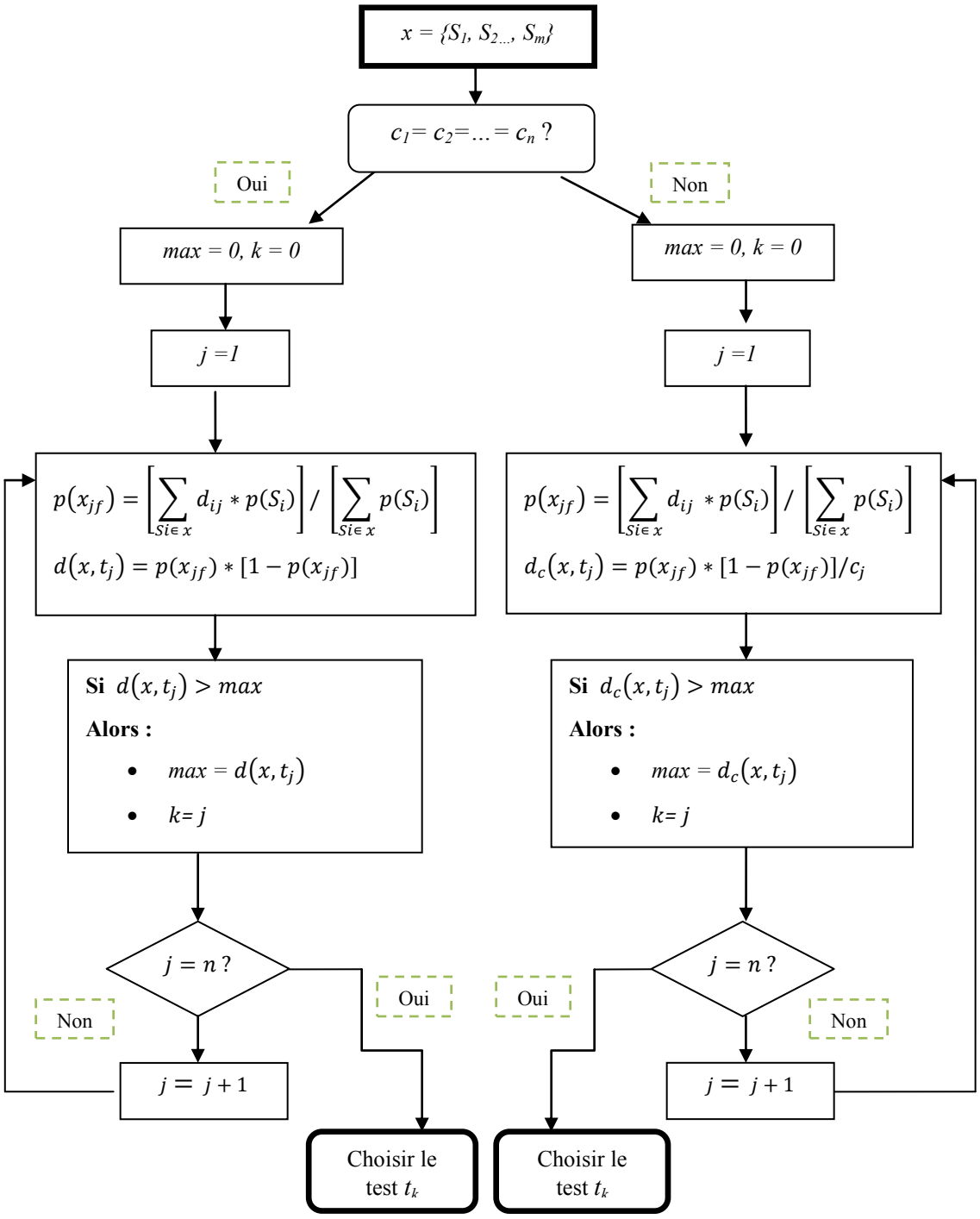

Figure 6. Algorithmes MSA et MSMCA

3.4. Application 1

On retourne à l'exemple 1 pour appliquer l'algorithme MSMCA:

- Pour $x = \{S_1, S_2\ldots, S_5\}$

Les tests disponibles sont T= $\{t_1, t_2\ldots, t_5\}$

Pour le test t_1

$$p(x_{1f}) = \left[\sum_{Si\in x} d_{i1} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.45$$

$$d_c(x, t_1) = p(x_{1f}) * [1 - p(x_{1f})]/c_1 = 0.45 * 0.55/6.5 = 0.038$$

Pour le test t_2

$$p(x_{2f}) = \left[\sum_{Si\in x} d_{i2} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.55$$

$$d_c(x, t_2) = p(x_{2f}) * [1 - p(x_{2f})]/c_2 = 0.55 * 0.45/5 = 0.0495$$

Pour le test t_3

$$p(x_{3f}) = \left[\sum_{Si\in x} d_{i3} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.4$$

$$d_c(x, t_3) = p(x_{3f}) * [1 - p(x_{3f})]/c_3 = 0.4 * 0.6/4 = 0.06$$

Pour le test t_4

$$p(x_{4f}) = \left[\sum_{Si\in x} d_{i4} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.6$$

$$d_c(x, t_4) = p(x_{4f}) * [1 - p(x_{4f})]/c_4 = 0.6 * 0.4/3 = 0.08$$

Pour le test t_5

$$p(x_{5f}) = \left[\sum_{Si \in x} d_{i5} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.5$$

$$d_c(x, t_5) = p(x_{5f}) * [1 - p(x_{5f})]/c_5 = 0.5 * 0.5/6 = 0.041$$

$max\, d_c(x, t_j \in T) = 0.08$ qui correspond au test t_4, alors, ce test sera choisi pour séparer le nœud x = {S_1, S_2…, S_5} (voir Figure 7).

Pour simplifier le traçage de l'arbre de décision, dans la suite de ce mémoire, chaque branche à gauche va correspondre à l'échec du test alors que celle de droite va correspondre à son succès.

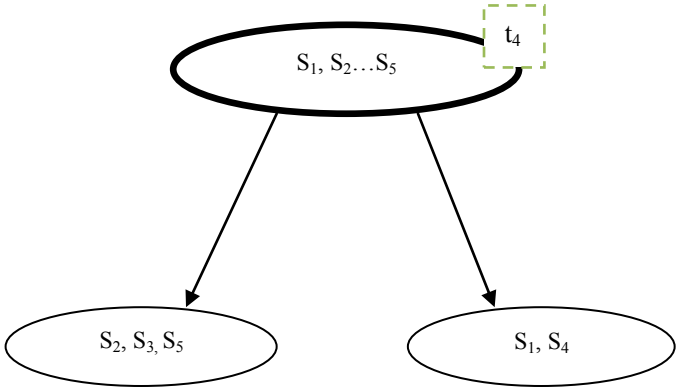

Figure 7. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme MSMCA

- Pour x = {S_2, S_3, S_5}:

Les tests disponibles sont T= {t_1, t_2…, t_5}

Pour le test t_1

$$p(x_{1f}) = \left[\sum_{Si \in x} d_{i1} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.583$$

$$d_c(x, t_1) = p(x_{1f}) * [1 - p(x_{1f})]/c_1 = 0.583 * 0.417/6.5 = 0.0374$$

Pour le test t_2

$$p(x_{2f}) = \left[\sum_{Si\in x} d_{i2} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.25$$

$$d_c(x, t_2) = p(x_{2f}) * [1 - p(x_{2f})]/c_2 = 0.25 * 0.75/5 = 0.0375$$

Pour le test t_3

$$p(x_{3f}) = \left[\sum_{Si\in x} d_{i3} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.66$$

$$d_c(x, t_3) = p(x_{3f}) * [1 - p(x_{3f})]/c_3 = 0.36 * 0.66/4 = 0.055$$

Pour le test t_4

$$p(x_{4f}) = \left[\sum_{Si\in x} d_{i4} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 1$$

$$d_c(x, t_4) = p(x_{4f}) * [1 - p(x_{4f})]/c_4 = 1 * 0/3 = 0$$

Pour le test t_5

$$p(x_{5f}) = \left[\sum_{Si\in X} d_{i5} \ * p(S_i)\right] / \left[\sum_{Si\in X} p(S_i)\right] = 0.33$$

$$d_c(x, t_5) = p(x_{5f}) * [1 - p(x_{5f})]/c_5 = 0.33 * 0.66/6 = 0.037$$

$max\ d_c(x, t_j \in T) = 0.055$ qui correspond au test t_3, alors, ce test sera choisi pour séparer le nœud x = {S_1, S_3, S_5} (voir Figure 8).

- Pour le nœud x = {S_1, S_4} :

Les tests qui permettent de séparer les causes de défaillance {S_1, S_4} sont t_1 et t_5. Le nœud contient deux causes de défaillance, alors vérifie la même séparation pour les deux tests t_1 et t_5, on choisit en ce moment le test le moins coûteux et ça sera t_5 (voir Figure 8).

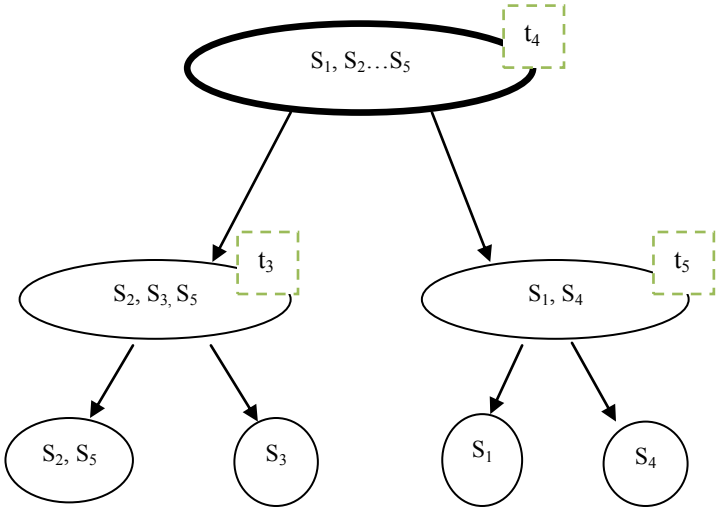

Figure 8. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme MSMCA (2)

- Pour x = {S_2, S_5}

Les tests qui permettent de séparer les causes de défaillance {S_2, S_5} sont t_1 et t_2. Le nœud contient deux causes de défaillance, alors il vérifie la même séparation pour les deux tests t_1 et t_2, on choisit en ce moment le test le moins coûteux et ça sera t_2 (voir Figure 9).

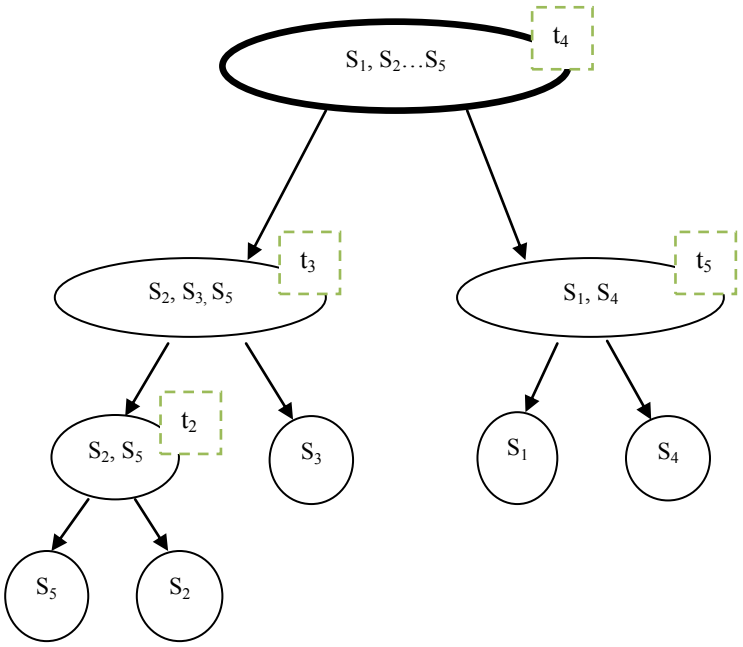

Figure 9. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme MSMCA (3)

Coût total prévu :

$cp_1 = p_1 * (c_4 + c_5)$ *(*cp_1 est le coût de tests encouru pour identifier la cause S_1)

$cp_2 = p_2 * (c_4 + c_3 + c_2)$

$cp_3 = p_3 * (c_4 + c_3)$

$cp_4 = p_4 * (c_5 + c_4)$

$cp_5 = p_5 * (c_3 + c_2 + c_4)$

Le coût total prévu est la somme des coûts encourus par tous les chemins formant l'arbre de décision :

$CT = cp_1 + cp_2 + cp_3 + cp_4 + cp_5$

$CT = p_1 * (c_4 + c_5) + p_2 * (c_4 + c_3 + c_2) + p_3 * (c_4 + c_3) + p_4 * (c_5 + c_4)$

$+ p_5 * (c_3 + c_2 + c_4)$

$CT = 9.8$

On remarque que 9.8 < 11.57. Le nouvel algorithme donne de meilleurs résultats que l'arbre généré avec le choix aléatoire de tests.

Remarque

Dans les travaux de GAO [8], le gain d'information rapporté par le test t_j est évalué par la fonction $d\left(x, t_j\right)$ ou $d_c\left(x, t_j\right)$. Avec d'autres chercheurs comme GAREY et GRAHAM [9], lors de la formation de l'arbre de décision, un test t_j est choisi s'il maximise $k(x, t_j)$ tel que :

$$k(x, t_j) = \frac{IG\left(x, t_j\right)}{c_j} \tag{4.19}$$

avec :

$$IG(x, t_j) = -[p(x_{jp}) * log_2(x_{jp}) + p(x_{jf}) * log_2(x_{jf})] \quad (4.20)$$

On parle en ce moment de l'algorithme (IG).

3.5. Application 2

On retourne à l'exemple 1 pour appliquer l'algorithme IG

- Pour x = {S_1, S_2…, S_5}

Les tests disponibles sont T= {t_1, t_2…, t_5}

Pour le test t_1

$$p(x_{1f}) = \left[\sum_{Si \in x} d_{i1} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.45$$

$$k(x, t_1) = -\frac{p(x_{1p}) * log_2(x_{1p}) + p(x_{1f}) * log_2(x_{1f})}{c_1} = 0.152$$

Pour le test t_2

$$p(x_{2f}) = \left[\sum_{Si \in x} d_{i2} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.55$$

$$d_c(x, t_2) = -\frac{p(x_{2p}) * log_2(x_{2p}) + p(x_{2f}) * log_2(x_{2f})}{c_2} = 0.196$$

Pour le test t_3

$$p(x_{3f}) = \left[\sum_{Si \in x} d_{i3} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.4$$

$$d_c(x, t_3) = -\frac{p(x_{3p}) * log_2(x_{3p}) + p(x_{3f}) * log_2(x_{3f})}{c_3} = 0.242$$

Pour le test t_4

$$p(x_{4f}) = \left[\sum_{Si \in x} d_{i4} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.6$$

$$d_c(x, t_4) = -\frac{p(x_{4p}) * log_2(x_{4p}) + p(x_{4f}) * log_2(x_{4f})}{c_4} = 0.323$$

Pour le test t_5

$$p(x_{5f}) = \left[\sum_{Si \in x} d_{i5} \ * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right] = 0.5$$

$$d_c(x, t_5) = -\frac{p(x_{5p}) * log_2(x_{5p}) + p(x_{5f}) * log_2(x_{5f})}{c_5} = 0.16$$

$max\, d_c(x, t_j \in T) = 0.323$ qui correspond au test t_4, alors, ce test sera choisi pour séparer le nœud x = {S_1, S_2…, S_5} (voir Figure 10).

Pour simplifier le traçage de l'arbre de décision, dans la suite de ce mémoire, chaque branche à gauche va correspondre à l'échec du test alors que celle de droite va correspondre à son succès.

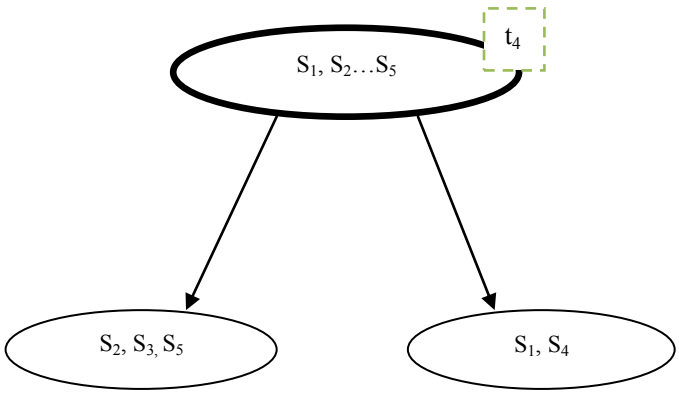

Figure 10. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme IG

- Pour x = {S_2, S_3, S_5}

Les tests disponibles sont T= {t_1, t_2…, t_5}

Pour le test t_1

$$p(x_{1f}) = \left[\sum_{Si\in x} d_{i1} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.583$$

$$d_c(x, t_1) = -\frac{p(x_{1p}) * log_2(x_{1p}) + p(x_{1f}) * log_2(x_{1f})}{c_1} = 0.125$$

Pour le test t_2

$$p(x_{2f}) = \left[\sum_{Si\in x} d_{i2} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.25$$

$$d_c(x, t_2) = -\frac{p(x_{2p}) * log_2(x_{2p}) + p(x_{2f}) * log_2(x_{2f})}{c_2} = 0.162$$

Pour le test t_3

$$p(x_{3f}) = \left[\sum_{Si\in x} d_{i3} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 0.66$$

$$d_c(x, t_3) = -\frac{p(x_{3p}) * log_2(x_{3p}) + p(x_{3f}) * log_2(x_{3f})}{c_3} = 0.230$$

Pour le test t_4

$$p(x_{4f}) = \left[\sum_{Si\in x} d_{i4} \ * p(S_i)\right] / \left[\sum_{Si\in x} p(S_i)\right] = 1$$

$$d_c(x, t_4) = -\frac{p(x_{4p}) * log_2(x_{4p}) + p(x_{4f}) * log_2(x_{4f})}{c_4} = 0$$

Pour le test t_5

$$p(x_{5f}) = \left[\sum_{Si \in X} d_{i5} \ * p(S_i)\right] / \left[\sum_{Si \in X} p(S_i)\right] = 0.33$$

$$d_c(x, t_5) == -\frac{p(x_{5p}) * log_2(x_{5p}) + p(x_{5f}) * log_2(x_{5f})}{c_5} = 0.153$$

$max\ d_c(x, t_j \in T) = 0.230$ qui correspond au test t_3, alors, ce test sera choisi pour séparer le nœud x = {S_1, S_3, S_5} (voir Figure 11).

- Pour le nœud x = {S_1, S_4}

Les tests qui permettent de séparer les causes de défaillance {S_1, S_4} sont t_1 et t_5. Le nœud contient deux causes de défaillance, alors vérifie la même séparation pour les deux tests t_1 et t_5, on choisit en ce moment le test le moins coûteux et ça sera t_5 (voir Figure 11).

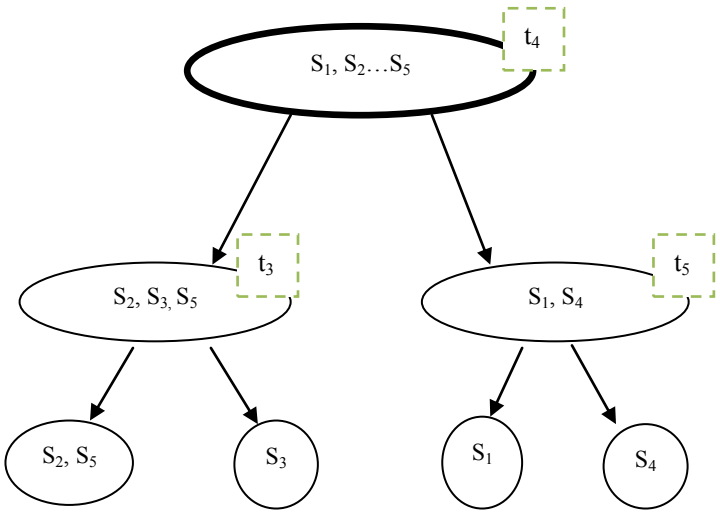

Figure 11. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme IG (2)

- Pour $x = \{S_2, S_5\}$:

Les tests qui permettent de séparer les causes de défaillance $\{S_2, S_5\}$ sont t_1 et t_2. Le nœud contient deux causes de défaillance, alors il vérifie la même séparation pour les deux tests t_1 et t_2, on choisit en ce moment le test le moins coûteux et ça sera t_2 (voir Figure 12).

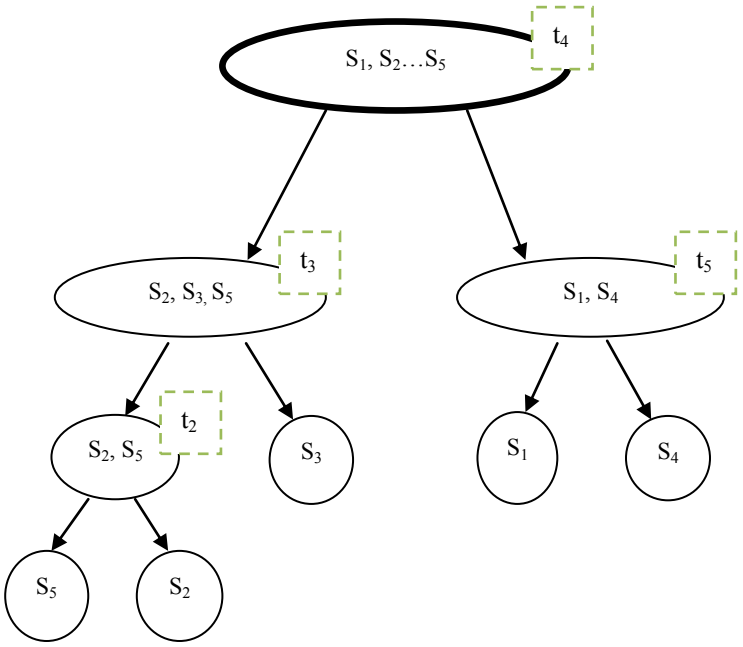

Figure 12. Génération de l'arbre de décision de l'exemple 1 en appliquant l'algorithme IG (3)

Remarque

Pour notre exemple, l'arbre généré en utilisant l'algorithme IG est le même que l'arbre généré en utilisant l'algorithme MSMCA.

3.6. Algorithme basé sur la programmation dynamique

Principe de la programmation dynamique (PD)

L'algorithme DP a été proposé par PATTIPATI en 1990 [21]. Cet algorithme se base sur les résultats de l'arbre de décision et l'information recueillie à partir du test effectué. Pour

chaque nœud de l'arbre, l'algorithme DP applique toutes les possibilités de tests. Une fois l'arbre est généré au complet, on utilise une approche ascendante commençant des nœuds terminaux vers le nœud origine. Tout le long de ce chemin, chaque fois qu'on est face à un nœud x où plusieurs choix de tests ont été appliqués, on doit sélectionner celui qui minimise la fonction $h^*(x)$ tel que :

$$h^*(x) = min_j\{c_j + h^*(x, t_j)\} \tag{4.21}$$

sachant que

- c_j est le coût encouru lors du test t_j
- $h^*(x, t_j)$ est le coût encouru en appliquant le test t_j au nœud x:

$$h^*(x, t_j) = p(x_{jp}) * h^*(x_{jp}) + p(x_{jf}) * h^*(x_{jf}) \tag{4.22}$$

avec :

$x_{jf} = \{S_i |\, d_{ij} = 1,\ S_i \in x\}$

$x_{jp} = \{S_i |\, d_{ij} = 0,\ S_i \in x\}$

$$p(x_{jf}) = \left[\sum_{Si \in x} d_{ij} * p(S_i)\right] / \left[\sum_{Si \in x} p(S_i)\right]$$

$p(x_{jp}) = 1 - p(x_{jf})$

- $h^*(S_i) = 0$ pour tout $0 \leq i \leq m$

(4.23)

Remarque

Lors de la construction de l'arbre, les algorithmes MSA et MSMCA choisissent les tests qui maximisent la fonction $d(x, t_j)$ ou $d_c(x, t_j)$. L'approche descendante utilisée est moins complexe car elle ne considère pas toutes les possibilités de tests contrairement à la

programmation dynamique. Un exemple tiré de l'article de PATTIPATI [21] sera repris à la section 4.5 pour comparer les résultats générés par ces deux méthodes.

3.7. Exemple 2

On propose le dictionnaire de tests du Tableau 6 tiré de l'article de PATTIPATI [21]. Les coûts de tests sont choisis égaux pour faciliter les calculs.

Tableau 6 : Dictionnaire de tests de l'exemple 2

Causes de défaillance \ *Tests*	t_1	t_2	t_3	t_4	t_5	*Probabilités de défaillance*
S_0	0	0	0	0	0	**0.70**
S_1	0	1	0	0	1	**0.01**
S_2	0	0	1	1	0	**0.02**
S_3	1	0	0	1	1	**0.10**
S_4	1	1	0	0	0	**0.05**
S_5	1	1	1	1	0	**0.12**
Le coût du test	1	1	1	1	1	**$\sum = 1$**

3.7.1. Résolution du problème en utilisant l'algorithme DP

Première étape :

On construit le nœud origine formé de toutes les causes de défaillances. Pour séparer ce nœud, on applique tous les tests du Tableau 6 (voir Figure 13)

Deuxième étape :

On utilise tous les tests disponibles permettant la séparation des nouveaux nœuds formés. La procédure est répétée jusqu'à atteindre les nœuds terminaux. Comme illustration, on va juste développer le nœud $\{S_0, S_1, S_2\}$ (voir Figure 14)

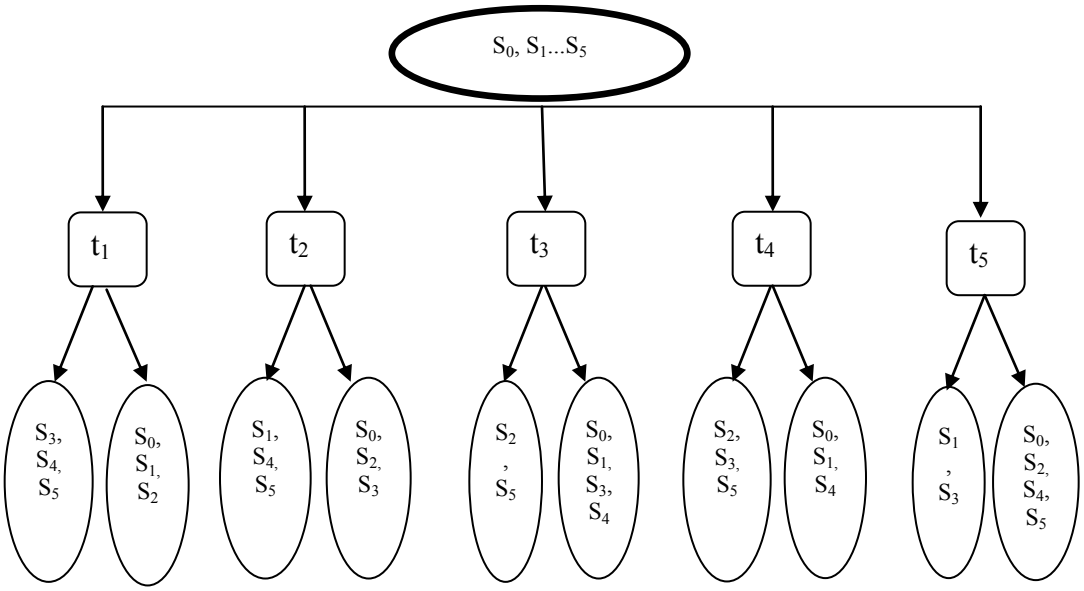

Figure 13. Génération de la séquence de tests en utilisant l'algorithme DP

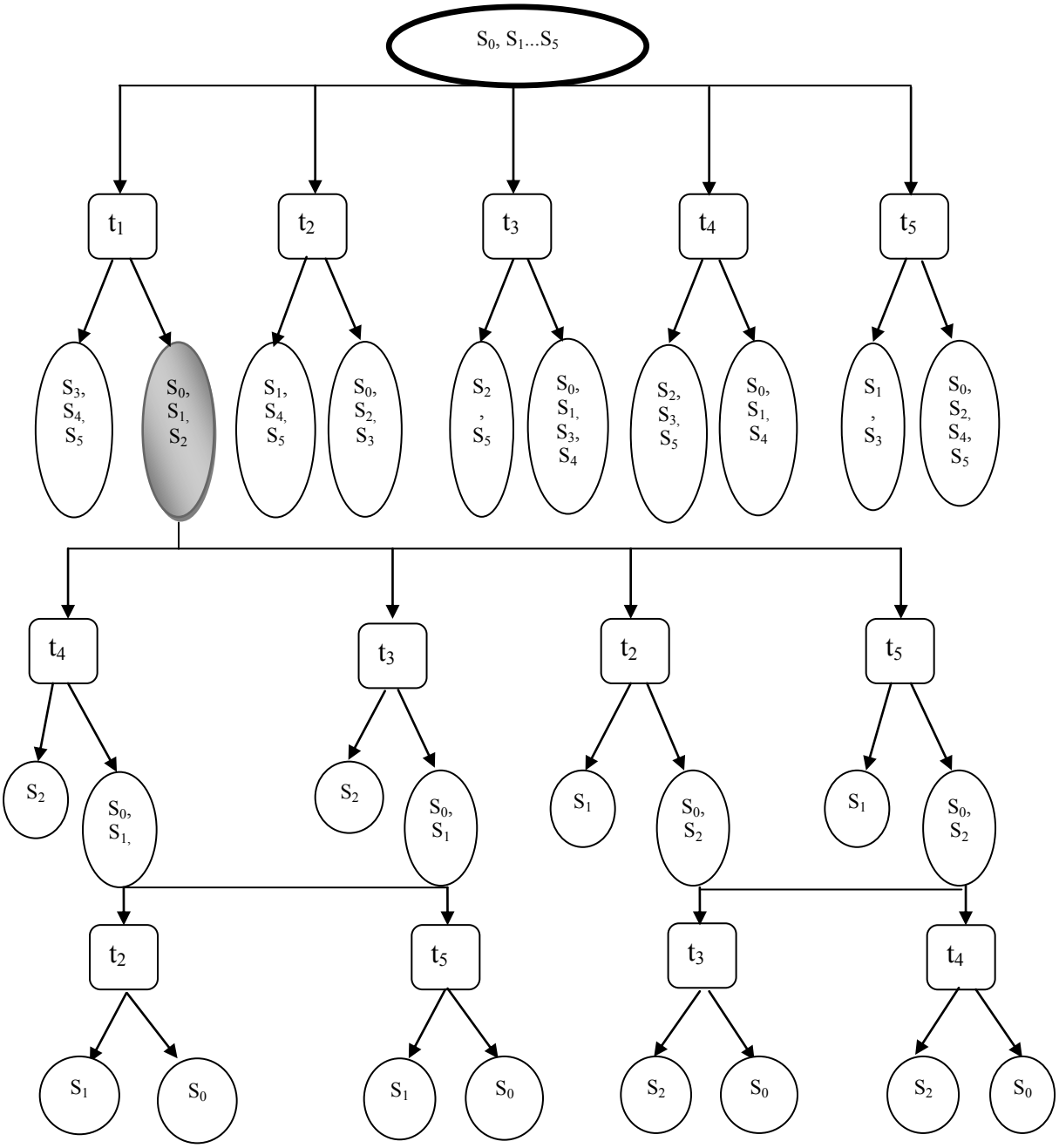

Figure 14. Génération de la séquence de tests en utilisant l'algorithme DP (2)

Troisième étape

On calcule la fonction $h^*(x)$ pour chaque nœud x de l'arbre de décision en commençant par les nœuds terminaux (les résultats des calculs sont placés dans un carré en pointillé sur la Figure 15)

❖ Pour le nœud $x = \{S_0, S_1\}$. Les tests qui permettent de séparer ce nœud sont: t_2 ou t_5.

Pour ces tests

- $x_{2p} = x_{5p} = S_0$
- $x_{2f} = x_{5f} = S_1$
- $h^*(x = \{S_0, S_1\}, t_2) = p(x_{2p}) * h^*(x_{2p}) + p(x_{2f}) * h^*(x_{2f})\}$
- $h^*(x = \{S_0, S_1\}, t_5) = p(x_{5p}) * h^*(x_{5p}) + p(x_{5f}) * h^*(x_{5f})\}$

d'où

- $h^*(x = \{S_0, S_1\}, t_2) = 0$ car $h^*(x_{2f}) = h^*(S_1) = h^*(x_{2p}) = h^*(S_0) = 0$ d'après l'équation (4.23)
- $h^*(x = \{S_0, S_1\}, t_5) = 0$ car $h^*(x_{5f}) = h^*(S_1) = h^*(x_{5p}) = h^*(S_0) = 0$ d'après l'équation (4.23)

alors

$$h^*(x = \{S_0, S_1\}) = min_{j, j\epsilon(2,5)}\{c_j + h^*(x, t_j)\} = 1 + 0 = 1$$

Bien que le coût unitaire du test t_5 soit le même que celui t_2, t_2 est choisi pour cette étape.

❖ Dans un niveau plus haut, on calcule $h^*(x = \{S_0, S_1, S_2\}, t_j)_{j\epsilon(2,3,4,5)}$. Les tests qui permettent de séparer ce nœud sont: t_2, t_3, t_4 ou t_5

Pour le test t_3

$$h^*(x = \{S_0, S_1, S_2\}, t_3) = p(x_{3p}) * h^*(x_{3p} = \{S_0, S_1\}) + p(x_{3f}) * h^*(x_{3f} = S_2)$$
$$h^*(x = \{S_0, S_1, S_2\}, t_3) = (0.71/0.73) * 1 + 0 = 0.972$$

De même pour le test t_4

$h^*(x = \{S_0, S_1, s_2\}, t_4) = 0.972$

Pour le test t_5

$h^*(x_{5p} = \{S_0, S_2\}) = 1$

$h^*(x_{5f} = S_1) = 1$

alors

$h^*(x = \{S_0, S_1, S_2\}, t_5) = p(x_{5p}) * h^*(x_{5p} = \{S_0, S_2\}) + p(x_{5f}) * h^*(x_{5f} = S_1)\}$

$h^*(x = \{S_0, S_1, S_2\}, t_5) = \frac{0.72}{0.73} * 1 + 0 = 0.986$

De même pour le test t_2

$h^*(x = \{S_0, S_1, S_2\}, t_2) = 0.986$

ainsi

$h^*(x = \{S_0, S_1, S_2\}) = min_{t_j}\{c_j + h^*(x, t_j)\} = 1.972$ qui correspond aux tests t_3 et t_4.

Malgré que le coût unitaire du test t_3 soit le même que t_4, t_3 est choisi pour cette étape (voir Figure 15)

❖ Pour calculer $h^*(x = \{S_0, S_1, S_2, S_3, S_4, S_5\}, t_1)$, on doit déterminer le coût rapporté par le test du nœud $\{s_3, s_4, s_5\}$ (voir Figure 16)

Pour le nœud $x = \{S_0, S_1, S_2, S_3, S_4, S_5\}$ et le test t_1:

- $x_{1f} = \{S_3, S_4, S_5\}$
- $x_{1p} = \{S_0, S_1, S_2\}$
- $h^*(x = \{S_0, S_1, S_2, S_3, S_4, S_5\}, t_1) = p(x_{1p}) * h^*(x_{1p}) + p(x_{1f}) * h^*(x_{1f})$
- $h^*(x = \{S_0, S_1, S_2, S_3, S_4, S_5\}, t_1) = 0.73 * 1.97 + 0.27 * 1.56 = 1.86$

❖ On refait le même travail pour tous les tests séparant le nœud origine $x = \{S_0, S_1, S_2, S_3, S_4, S_5\}$. Le test t_2 sera choisi avec un coût total prévu égal 2.18. Comme illustration, on génère seulement l'arbre issu des tests t_1 et t_2 (voir Figure 16).

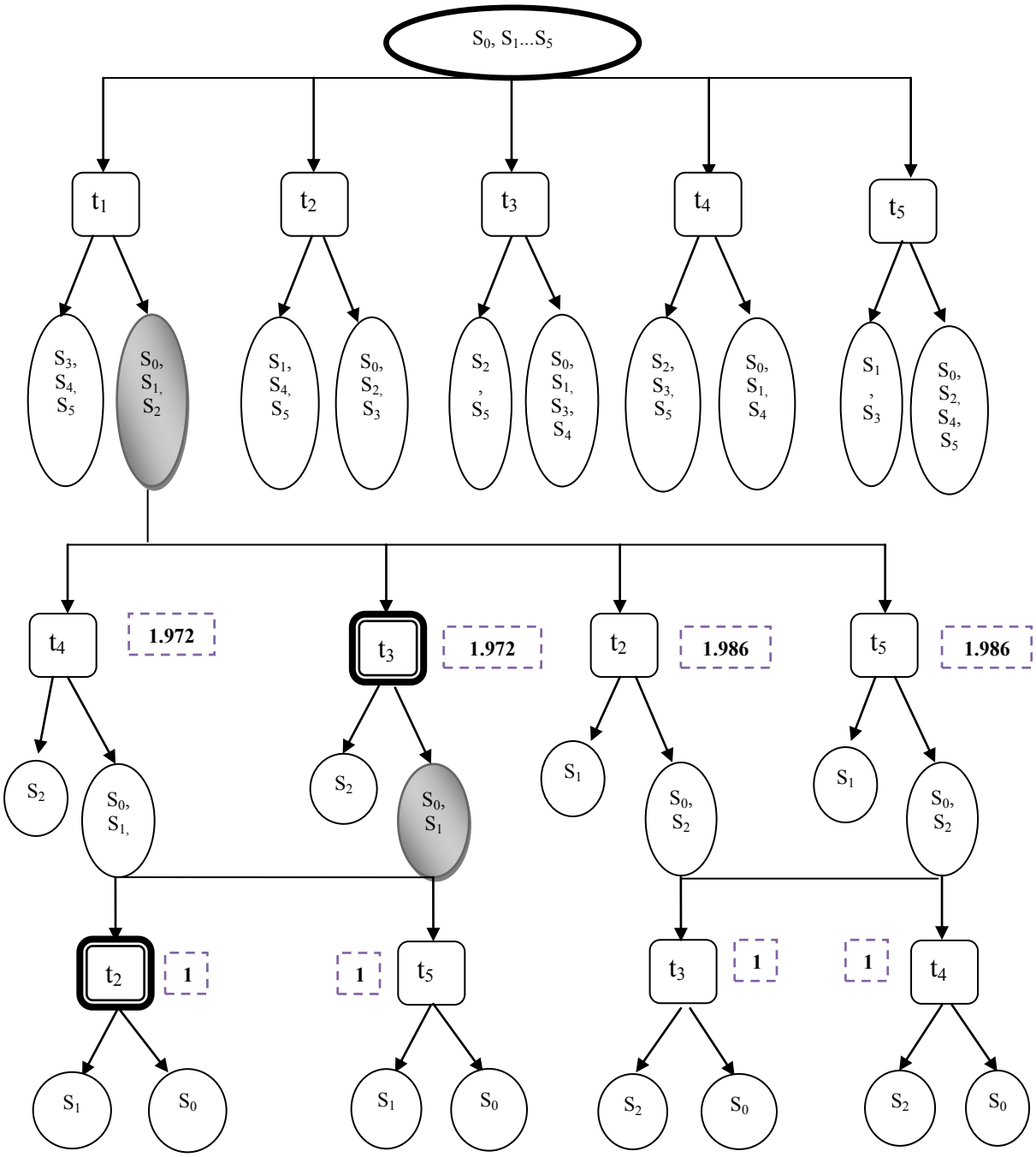

Figure 15. Génération de la séquence de tests en utilisant l'algorithme DP (3)

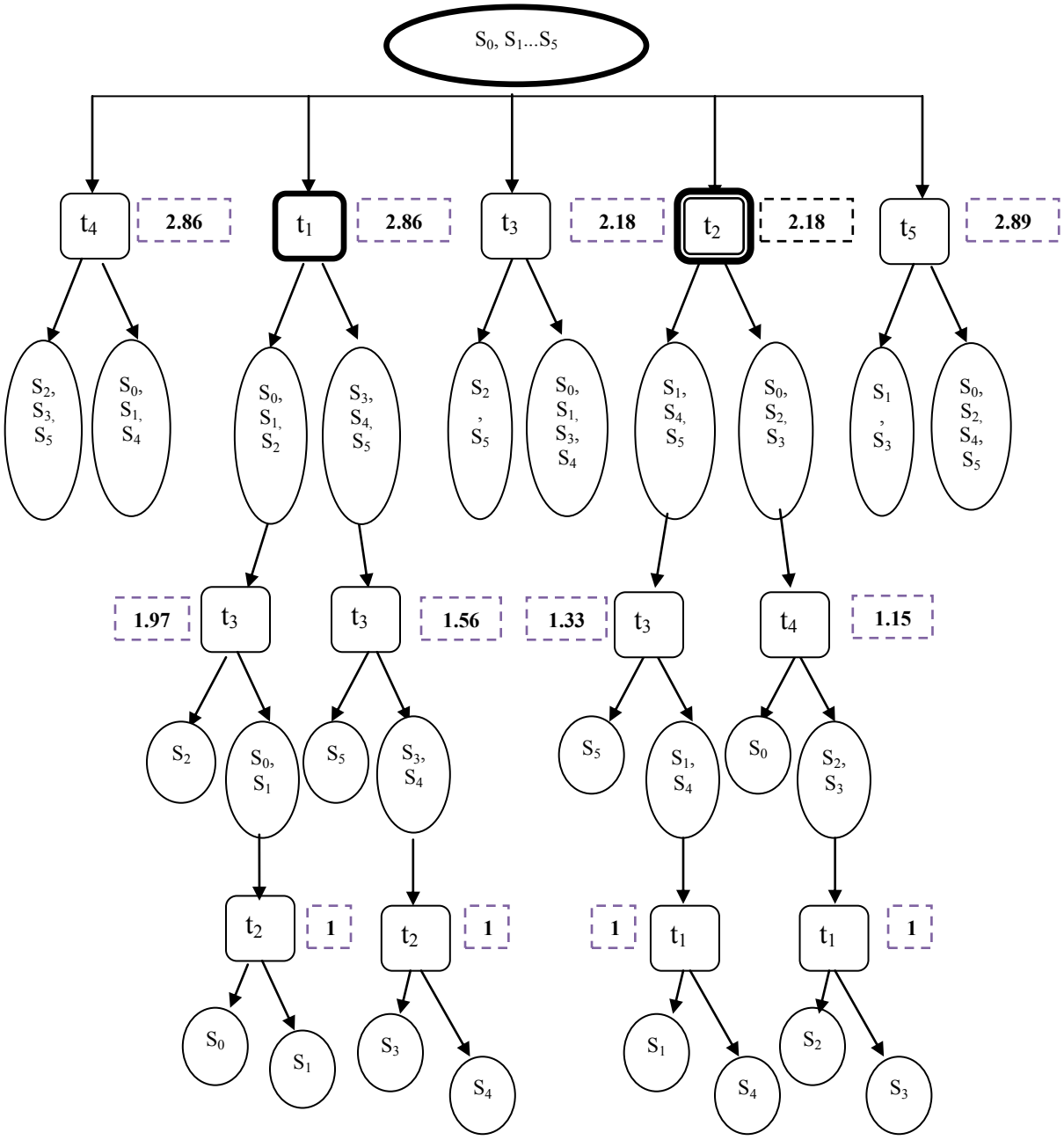

Figure 16. Génération de la séquence de tests en utilisant l'algorithme DP (4)

3.7.2. Résolution de l'exemple 2 en utilisant l'algorithme de séparation maximale

Utilisons l'algorithme de séparation maximale (MSA) pour traiter l'exemple 2. On ne va pas trop rentrer dans les calculs. Lors de l'application de la démarche descendante, le test choisi à chaque nœud est celui qui maximise l'information $d(x, t_j)$. Le test choisi pour séparer le nœud origine est t_1 avec $d(x, t_1) = 0.19$ ($d(x, t_2) = 0.14$). L'arbre de décision formé est présenté à la Figure 17.

Remarque

En utilisant l'algorithme MSA, la séquence de tests commence par t_1 et son coût total prévu est estimé à 2.83. Ce coût est supérieur à 2.18 correspondant à celui de la séquence de tests commençant par t_2 et généré par l'algorithme DP. L'algorithme (MSA) est simple à appliquer comparé à l'algorithme DP qui amène à des résultats plus satisfaisants en utilisant la démarche descendante qui regarde toutes les possibilités de tests jusqu'a ce que l'arbre complet est généré. Le problème de détection de panne peut être résolu en utilisant l'algorithme DP ou les algorithmes MSA, MSMCA et IG. Cependant, ces algorithmes ne produisent pas de bons résultats en présence d'un grand nombre d'états de défaillances [22]. Pour remédier à ce problème, il sera rentable de développer d'autres algorithmes heuristiques.

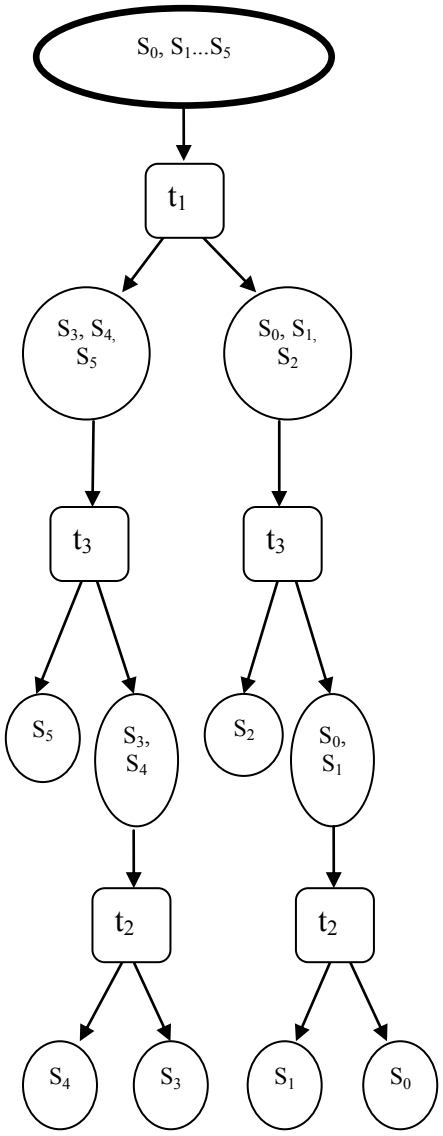

Figure 17. Génération de la séquence de tests en utilisant l'algorithme MSA

4. Les stratégies de déploiement

Les stratégies de déploiements sont apparues en 1996 avec BERTSEKAS et TSITSIKLIS. Ces stratégies sont souvent combinées aux principes de la théorie d'information et l'approche heuristique. Elles offrent une solution sous-optimale avec des économies considérables dans les calculs effectués. En se basant sur les travaux de PATTIPATI [21, 22], on va présenter les étapes de formation de l'algorithme RIG suivies d'une application sur l'exemple 2.

Algorithme RIG

Étape 1

- On crée le nœud origine $\bar{x}_l$
- On sépare l'ensemble des causes de défaillance du nœud origine en appliquant tous les tests t_j disponibles.

Étape 2

Étape 2.1 : Cette étape est présentée sur la Figure 18. PATTIPATI [22] présente ces étapes sous la forme d'un paragraphe très court et difficile à comprendre. On a essayé alors de rajouter quelques modifications reflétant l'aspect algorithmique de la démarche. Une structure itérative est mise en œuvre. Des conditions initiales et plusieurs paramètres sont posés.

Étape 2.2

Pour chaque nœud x_{jp} et x_{jf} directement issu du nœud origine $\bar{x}_l$ lors du test t_j, on calcule $h(x_{jp})$ et $h(x_{jf})$. Pour x_{ijp} :

$$h(x_{jp}) = \sum_{l=0}^{|\bar{x}|} \left\{ \sum_{k=1}^{|p_l|} cp_l\,[k] \right\} * p(\bar{x}_l) \tag{4.23}$$

Sachant que

- $\bar{x}$ est l'ensemble des nœuds terminaux atteints en décomposant le nœud x_{jp} issu du test t_j appliqué dans l'étape 1.
- $|\bar{x}|$ est le cardinal de $\bar{x}$
- p_l est la séquence de tests utilisés pour isoler le nœud x_{jp}
- $|p_l|$ est le cardinal de p_l

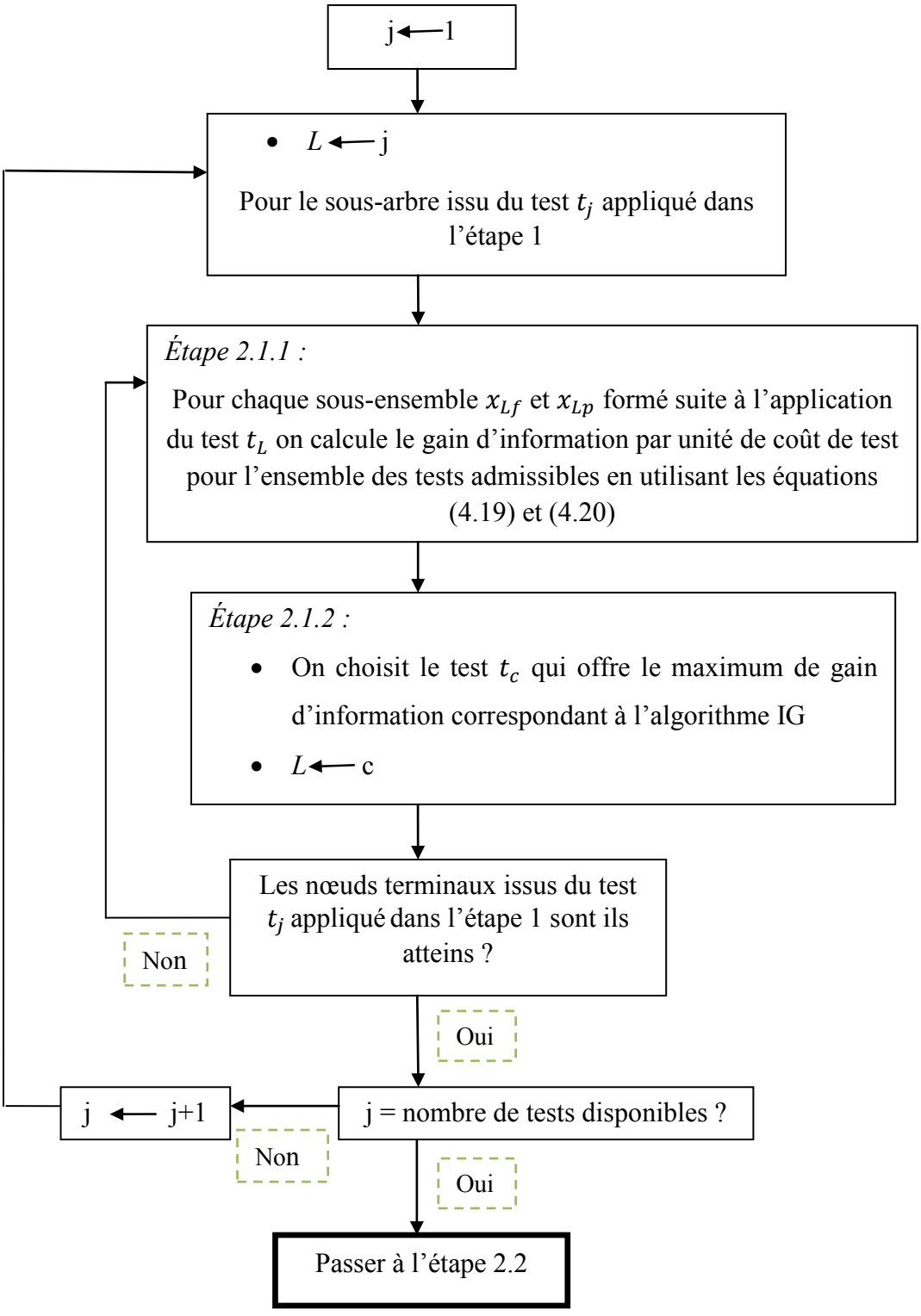

Figure 18. Étape 2.1 de l'algorithme RIG

Étape 2.3

On calcule le coût total prévu de chaque séquence de tests Se_j commençant par le test t_j appliqué dans la première étape.

$$h_{tj}(Se_j) = c_j + p(x_{jp}) * h(x_{jp}) + p(x_{jf}) * h(x_{jf}) \quad (4.24)$$

Étape 2.4

On choisit la séquence Se_j qui offre le coût minimal.

5. Application

Appliquons les étapes de l'algorithme RIG pour résoudre l'exemple 2 (Tableau 7)

Tableau 7 : Dictionnaire de tests de l'exemple 2

Causes de défaillance \ *Tests*	t_1	t_2	t_3	t_4	t_5	*Probabilités de défaillance*
S_0	0	0	0	0	0	**0.70**
S_1	0	1	0	0	1	**0.01**
S_2	0	0	1	1	0	**0.02**
S_3	1	0	0	1	1	**0.10**
S_4	1	1	0	0	0	**0.05**
S_5	1	1	1	1	0	**0.12**
Le coût du test	1	1	1	1	1	**∑ = 1**

Résolution

Au début et selon la démarche de PATTIPATI [22], on sépare l'ensemble des causes de défaillance du nœud origine en appliquant tous les tests t_j disponibles. On passe ensuite aux étapes 2.1.1 et 2.1.2 utilisant l'information heuristique. Pour le nœud $\{S_0, S_1, S_2\}$ issu du test t_1, le calcul utilisant les équations (4.19) et (4.20) montre que les tests t_3 et t_4 sont

ceux qui offrent le maximum de gain d'information par unité de coût tel que $k = 0.1812$. De leur côté, les tests t_2 et t_5 offrent un gain d'information égal à 0.104. Bien que le gain d'information du test t_4 soit le même que celui de t_3, t_3 est choisi pour cette itération. On continu le calcul jusqu'à atteindre les nœuds terminaux issus du nœud $\{S_0, S_1, S_2\}$. Le même travail est répété pour tous les sous-ensembles issus du nœud origine. Comme illustration, on schématise dans la Figue 19 les sous-arbres issus des tests t_1 et t_2 appliqués dans l'étape 1.

À l'étape 2.2, le coût de tests est calculé en utilisant l'équation (4.23). Pour le nœud $\{S_0, S_1, S_2\}$ issu du test t_1, le coût est évalué à 1.97 alors qu'il est égal à 1.56 pour le nœud $\{S_3, S_4, S_5\}$. Le coût total prévu du nœud commençant par le test t_1 est obtenu en utilisant l'équation (4.24) de l'étape 2.3 et est évalué à 2.86. La même procédure est faite pour les autres tests t_2, t_3, t_4 et t_5. Dans l'étape 2.4, on choisit le test qui offre le coût minimal. Le test t_2 est sélectionné avec un coût total prévu égal à 2.18.

Dans le cas où seulement l'information heuristique est utilisée, le test t_1 a été choisi avec un coût total prévu égal à 2.86 (voir Figure 19). Les stratégies de déploiement assurent en ce moment un compromis entre la simplicité d'exécution et le coût de tests offert.

Remarque

Les stratégies de déploiements offrent la séquence optimale de tests avec des économies considérables dans les calculs. Ces stratégies sont simples, caractérisées par une large gamme d'utilisation et une portée de mise en œuvre suffisamment étendue. Afin d'accélérer la stratégie de déploiement pour les systèmes complexes, PATTIPATI [22] a recommandé dans l'étape 2.1 de conserver uniquement les meilleurs (*E)* tests selon le gain d'information rapporté par unité de coût de test. Tant que $E > 5$, les résultats sont très proches de la solution optimale.

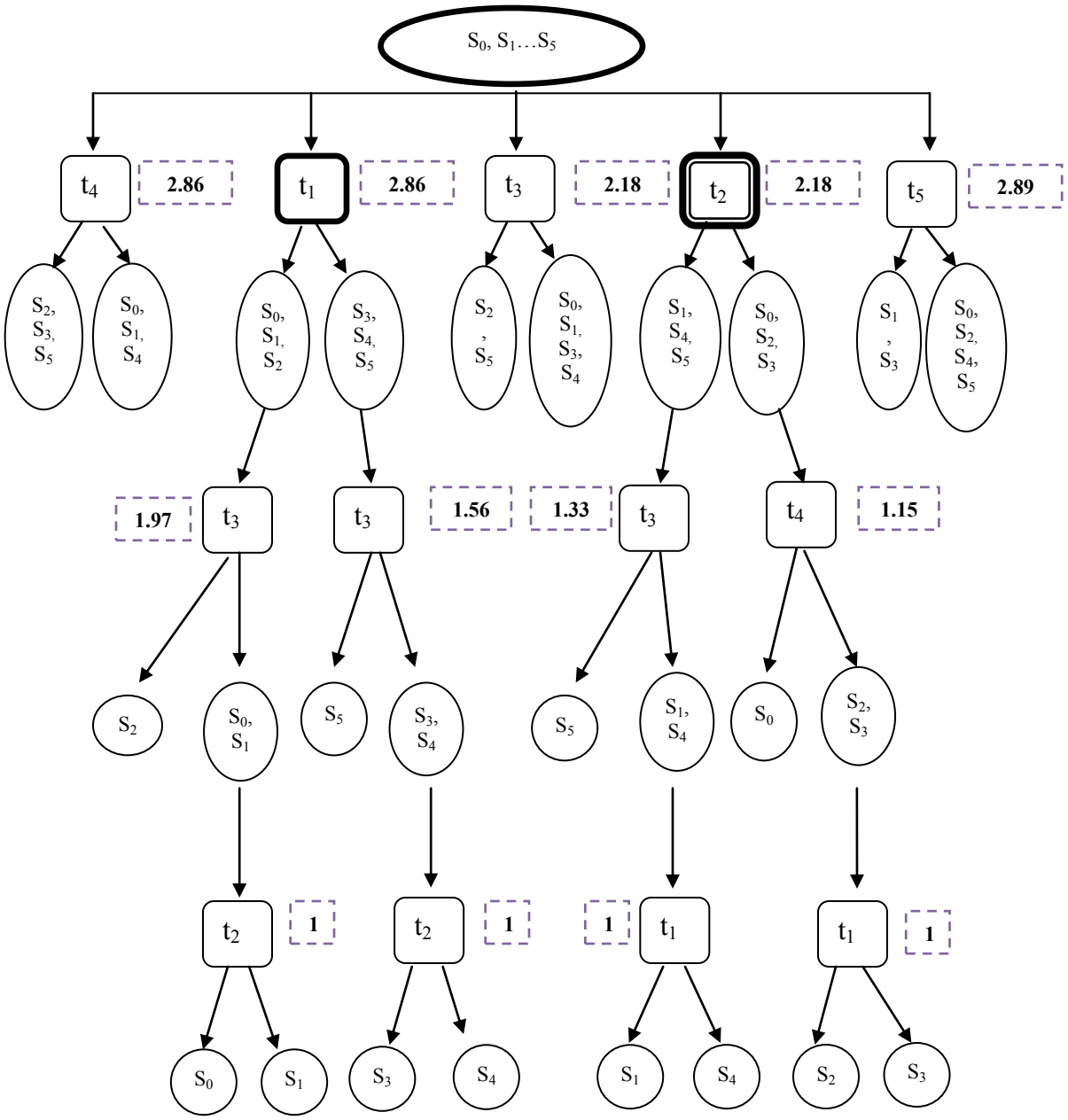

Figure 19. Génération de la séquence de tests en utilisant l'algorithme RIG

Chapitre 5 : Étude des systèmes complexes

1. Introduction

Le cas d'un système simple a été considéré dans les chapitres précédents. Plusieurs séquences de tests ont été mises en œuvre. Diverses approches ont été employées suivant l'information disponible et les limites imposées. Dans ce chapitre, on traite le cas des systèmes complexes où plusieurs composants peuvent tomber en panne simultanément. Plusieurs papiers ont été publiés sur le sujet. Ces travaux tentent d'exploiter l'information disponible pour la génération des séquences de tests en se basant sur le critère de minimisations de l'espérance mathématique du coût de tests. Les données utilisées sont, outre le diagramme de fiabilité du système, la probabilité de défaillance des composants et leur coût de test.

2. Définitions

2.1. Arbre de défaillance

La définition d'un arbre de défaillance est présentée dans le chapitre 2.

2.2. Diagramme de fiabilité

C'est un graphe muni d'une entrée et d'une sortie où tous les chemins sont formés d'une suite de composants et sont tels que « le système fonctionne si tous les composants d'au moins un chemin de ce graphe fonctionnent » [1], condition suffisante pour le fonctionnement d'un système.

2.3. Coupe minimale

Une coupe est une constitution d'événements élémentaires qui emmènent à un événement indésirable. Cette coupe est dite minimale si le manque d'au moins d'un des événements qui la construit, interdit l'apparition de l'événement recherché.

3. Séquence de tests en cas d'indépendance des coupes minimales

Selon la Figure 20, la réalisation de l'événement T est due à la survenance d'une des coupes minimales [1, 8]. La relation entre l'événement indésirable T et les coupes minimales peut être générée comme une structure reliant les M_i coupes connectées en série (voir Figure 21). La réalisation de l'une des coupes amène à l'événement redouté.

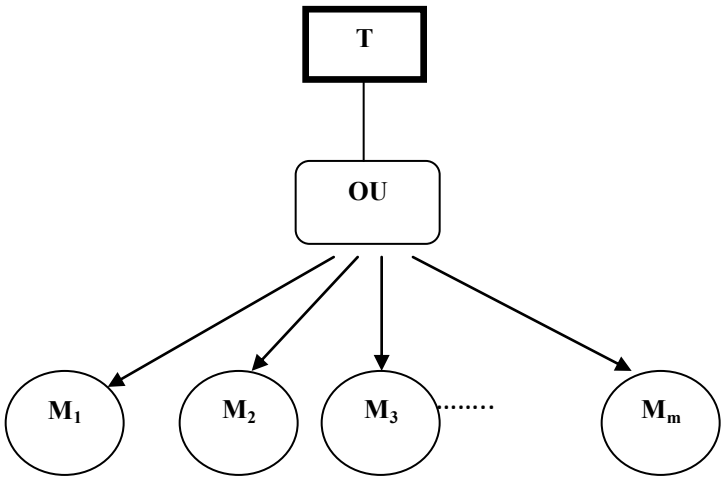

Figure 20. Arbre de défaillance et coupes minimales

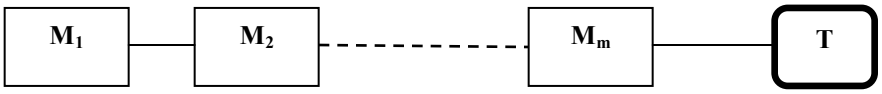

Figure 21. Système formé des coupes minimales connectées en série

En cas d'indépendance des coupes [1,8], les composants d'une coupe minimale n'appartiennent pas à d'autres coupes. La probabilité de défaillance d'une coupe minimale est le produit des probabilités de défaillances de ces composants. Le coût total moyen de chaque coupe est majoré par la somme des coûts de tests de l'ensemble de ces composants. Dans notre cas, les tests sont supposés parfaits. Il faut arrêter les tests lorsqu'on détecte la

coupe responsable de la défaillance. Une fois un composant testé est jugé en bon état, la coupe qui le contient est disculpée et on passe directement à la prochaine.

3.1. Ordonnancement optimal des coupes minimales

Le système est formé de (m) coupes minimales mises en série comme montre la figure 17. On veut trouver rapidement la coupe responsable de la défaillance tout en minimisant le coût de tests.

Théorème 5.1

Par analogie avec l'approche probabiliste, GAO et AIT KADI [8] prouvent que l'ordonnancement optimal des coupes minimales doit satisfaire les conditions suivantes:

$$\frac{p_j}{c_j} \geq \frac{p_{j+1}}{c_{j+1}} \qquad pour\ j \in [1..m-1] \tag{6.3}$$

$$c_{m-1} \leq c_m \tag{6.4}$$

Avec

- p_j est la probabilité conditionnelle que la coupe M_j soit responsable de la défaillance
- c_j est le coût de tests de la coupe M_j

Le coût total moyen de la séquence optimale de tests S_n^* est défini comme suit :

$$C(S_n^*) = E[C(S_n^*)] = c_1 + c_2 * (1 - p_1) + \cdots + C_k * \left(1 - \sum_{i=1}^{k-1} p_i\right) + C_{m-1} * \left(1 - \sum_{i=1}^{m-2} p_i\right) \tag{6.5}$$

3.2. Ordonnancement optimal des composants au sein de chaque coupe

Restons dans le cas où les coupes sont indépendantes. Pour trouver l'ordre dans lequel on teste les composants formant chaque coupe, il faut définir l'expression du coût de tests d'une coupe. En effet, on commence par tester le premier composant de la coupe M_j, s'il est en bon état, alors, on arrête le test pour passer à la prochaine. Sinon, si le composant 1 est jugé fautif on passe au test du composant 2. Le coût de test du composant 2 est alors multiplié par la probabilité p_{1j} que le composant 1 de la coupe M_j soit fautif. De même pour le composant 3 et ainsi de suite. L'expression du coût total moyen de la séquence de tests des composants appartenant à la même coupe minimale M_j est définie comme suit [1,8]:

$$C(S_j) = c_{1j} + c_{2j} * p_{1j} + c_{3j} * p_{1j} * p_{2j} + \cdots + c_{Ejj} * \prod_{k=1}^{Ej-1} p_{kj} \tag{6.6}$$

(E_j) est le nombre de composants contenant dans la coupe M_j . Pour avoir la séquence optimale de tests qui minimise $C(S_j)$, il sera avantageux de commencer par le composant le moins coûteux qui possède la plus grande probabilité d'être en bon état. En effet, si une coupe est responsable de la défaillance, tous ces composants doivent être défaillants. Sinon, si elle n'est pas la cause de la panne, il faut vite tomber sur le composant en bon état pour lâcher la coupe en question et passer à une autre.

Théorème 5.2

Selon GAO [8], pour chaque coupe, les composants doivent satisfaire la condition :

$$\frac{c_{ij}}{1-p_{ij}} \geq \frac{c_{ij+1}}{1-p_{ij+1}} \quad pour\ tout\ i = 1 \ldots E_j - 1 \tag{6.7}$$

Preuve

Soit :

- $S_j^* = [(1)(2) \dots (\mathrm{i}-1)(\mathrm{i})(\mathrm{i}+1) \dots (\mathrm{E_j})]$ est la séquence optimale obtenue en ordonnant les composants dans l'ordre croissant de $\frac{c_{ij}}{1-p_{ij}}$
- $S_j^{'} = [(1)\ (2) \dots (\mathrm{i}-1)(\mathrm{i}+1), (\mathrm{i}) \dots (\mathrm{E_j})]$ est la séquence obtenue en permutant la position des deux composants (i) et (i+1) dans la séquence S_j^*

On calcule la différence de coût entre les deux séquences S_j^* et $S_j^{'}$

$$C(S_j^*) = c_{1j} + c_{2j} * p_{1j} + c_{3j} * p_{1j} * p_{2j} + \cdots + c_{ij} * \prod_{k=1}^{i-1} p_{kj} + c_{(i+1)j} * \prod_{k=1}^{i} p_{kj} + \cdots + c_{Ejj} * \prod_{k=1}^{Ej-1} p_{kj} \tag{6.8}$$

$$C(S_j^*) - c(S_j^{'}) = [c_{ij} * \prod_{k=1}^{i-1} p_{kj} + c_{(i+1)j} * \prod_{k=1}^{i-1} p_{kj}] - [c_{(i+1)j} * \prod_{k=1}^{i-1} p_{kj} + c_{ij} * \prod_{k=1}^{i-1} p_{kj} - p_{(i+1)j}] \tag{6.9}$$

$$C(S_j^*) - c(S_j^{'}) = [c_{ij} * (1 - p_{(i+1)j}) - c_{(i+1)j}(1 - p_{ij})] * \prod_{k=1}^{i-1} p_{kj} \tag{6.10}$$

Pour que S_j^* soit la séquence optimale sachant que $\prod_{k=1}^{i-1} p_{kj} > 0$, il faut que $C(S_j^*) - C(S_j^{'}) < 0$, ce qui est équivalent à dire :

$c_{ij} * (1 - p_{(i+1)j}) - c_{(i+1)j} * (1 - p_{ij}) < 0$ (Condition vérifiée)

Pour résumer la démarche, on présente dans la Figure 22 la procédure générale pour la génération de la séquence optimale de tests en cas d'indépendance des coupes minimales.

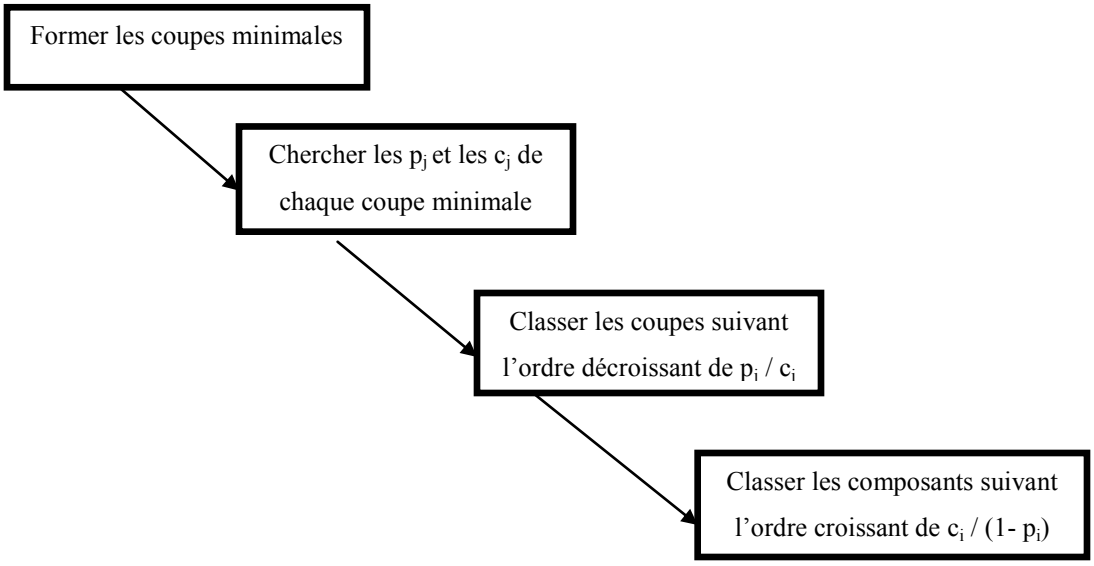

Figure 22. Procédure générale pour la génération de la séquence optimale de tests en cas d'indépendance des coupes minimales [1]

4. Séquence de tests en cas de dépendance des coupes minimales

Si on écarte l'hypothèse d'indépendance des coupes minimales, alors, pour un composant qui est testé et jugé en bon état, chaque coupe minimale qui le contient ne sera pas considérée et son coût de tests sera nul. De plus, si un composant est déjà testé et jugé fautif, alors il faut ne pas en tenir compte dans les prochaines coupes minimales. Ces modifications peuvent changer l'ordre des coupes minimales non testées et classées suivant le rapport p/c. Il est nécessité en ce moment de refaire les calculs pour réordonner les coupes. Pour une coupe minimale soit Φ l'ensemble des composants qui n'ont pas été

testés. Suivant GAO et AIT-KADI [8] et PORTMANN [1], la probabilité de défaillance conditionnelle de la coupe minimale est :

$$p_k = \prod_{i \epsilon \Phi} p_{ik} \quad pour\ k = 1,2, \dots m \tag{6.11}$$

Le coût de tests de cette coupe est:

$$C_k = \sum_{i \epsilon \Phi} c_{ik} \tag{6.12}$$

La Figure 23 présente la démarche à suivre pour la génération de la séquence de tests en cas de dépendance des coupes minimales.

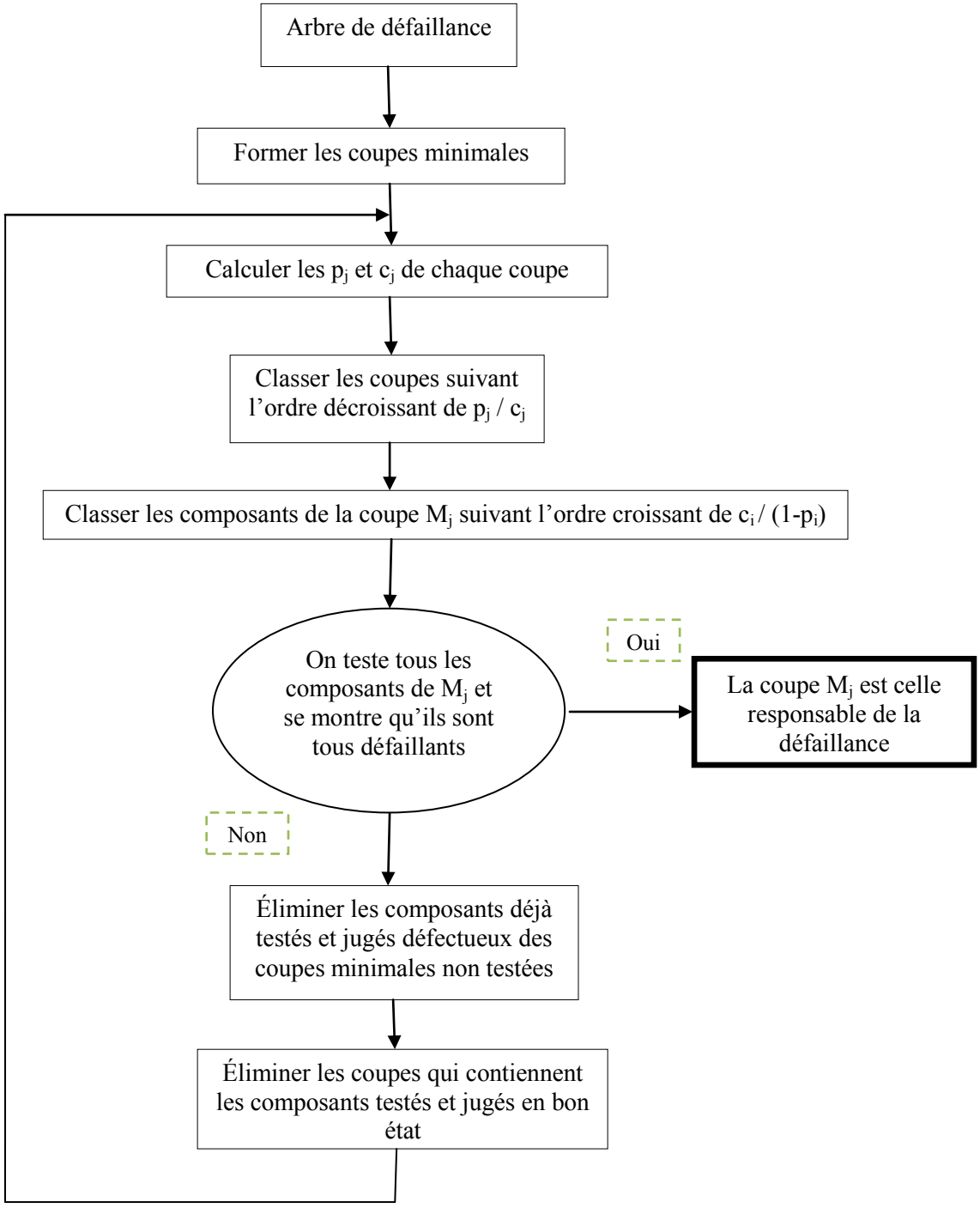

Figure 23. Procédure détaillée pour la génération de la séquence de tests en cas de dépendance des coupes minimales

5. Exemple 3

On propose l'exemple tiré de la thèse de GAO [8]. L'arbre de défaillance est présenté dans la Figure 24. Cet arbre contient 5 coupes minimales et 10 composants. Les paramètres sont présentés dans le Tableau 8. e_{ik} est le composant e_i se trouvant dans la coupe M_k. Supposons que tous les composants ont des probabilités de défaillance égales à 0.1. Les coûts de tests des composants sont: $c_1 = c_2 = c_3 = 2.5$, $c_4 = c_5 = c_8 = 3.5$, $c_6 = c_7 = 4$, $c_9 = c_{10}$ $c_{10} = 2$.

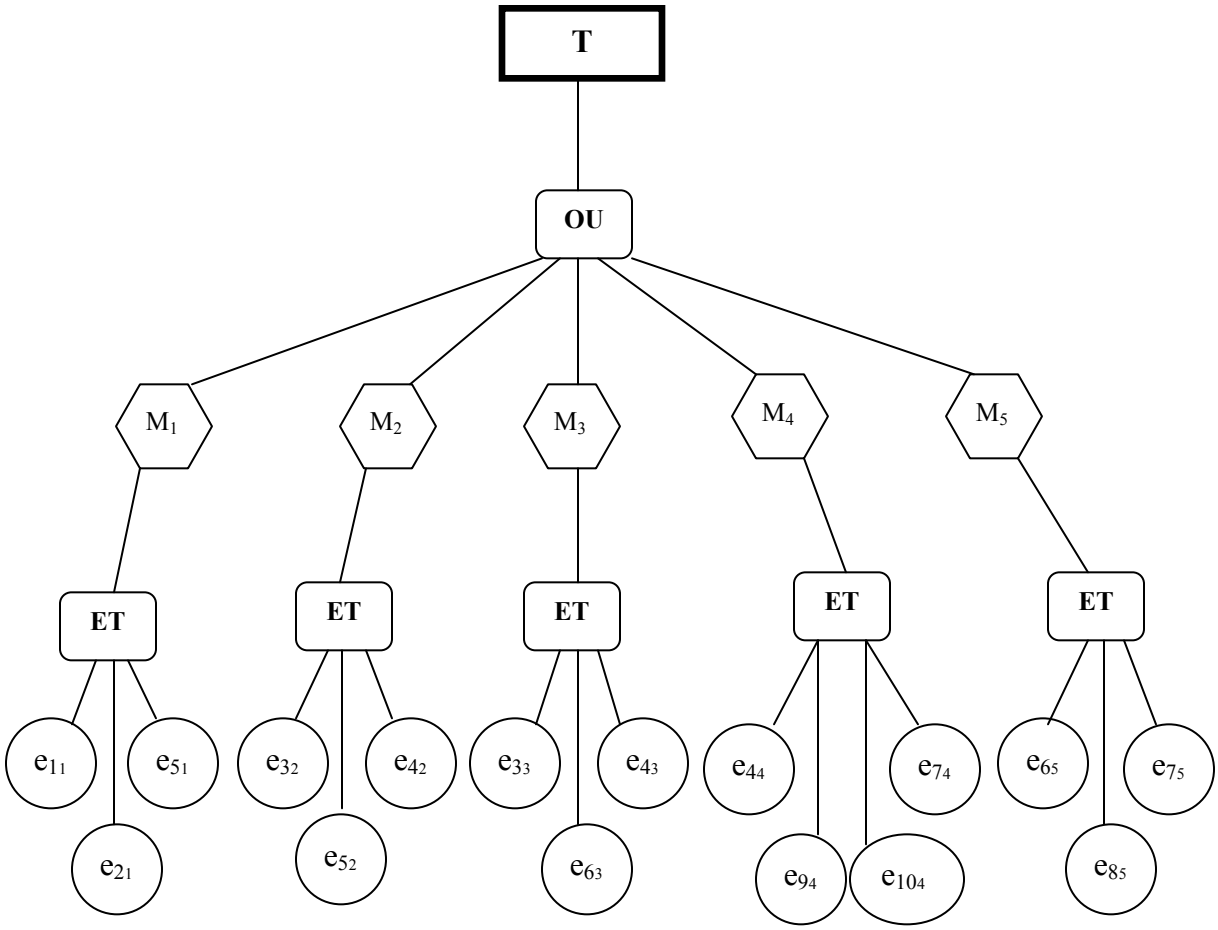

Figure 24. Arbre de défaillance de l'exemple 3

Tableau 8 : Paramètres du système de l'exemple 3

Coupe minimale M_j	Composants formant la coupe minimale M_j	Probabilité p_j	Coût de tests c_j	Le rapport p_j/c_j
1	e_{11}, e_{21}, e_{51}	0.001	8.5	$1.18\ 10^{-4}$
2	e_{32}, e_{42}, e_{52}	0.001	9.5	$1.05\ 10^{-4}$
3	e_{32}, e_{42}, e_{62}	0.001	10	10^{-4}
4	$e_{44}, e_{74}, e_{94}, e_{104}$	0.0001	11.5	$8.69\ 10^{-6}$
5	e_{65}, e_{75}, e_{85}	0.001	11.5	$8.69\ 10^{-6}$

En classant les coupes minimales suivant le rapport p_j/c_j (Probabilité de défaillance conditionnelle de la coupe M_j / Coût de tests de la coupe M_j), la coupe M_1 occupe la première position avec le plus grand rapport $p_1/c_1 = 1.18\ 10^{-4}$. On classe ensuite les composants de la coupe M_1 suivant le rapport $c_i/(1-p_i)$. Les calculs donnent $c_1/(1-p_1) = 2.78$, $c_2/(1-p_2) = 2.78$ et $c_5/(1-p_5) = 3.89$. Ainsi, on commence à tester le composant e_{11} puis e_{21} et enfin e_{51}. Les figures qui suivent présentent les démarches pour retrouver la coupe responsable de la défaillance [8]. On rappelle que tout composant testé et jugé fautif ne sera plus considéré dans les coupes prochaines qui le contiennent. Par contre, si ce composant est en bon état, les coupes minimales qui le contiennent doivent être éliminées, car elles ne sont pas responsables de la défaillance.

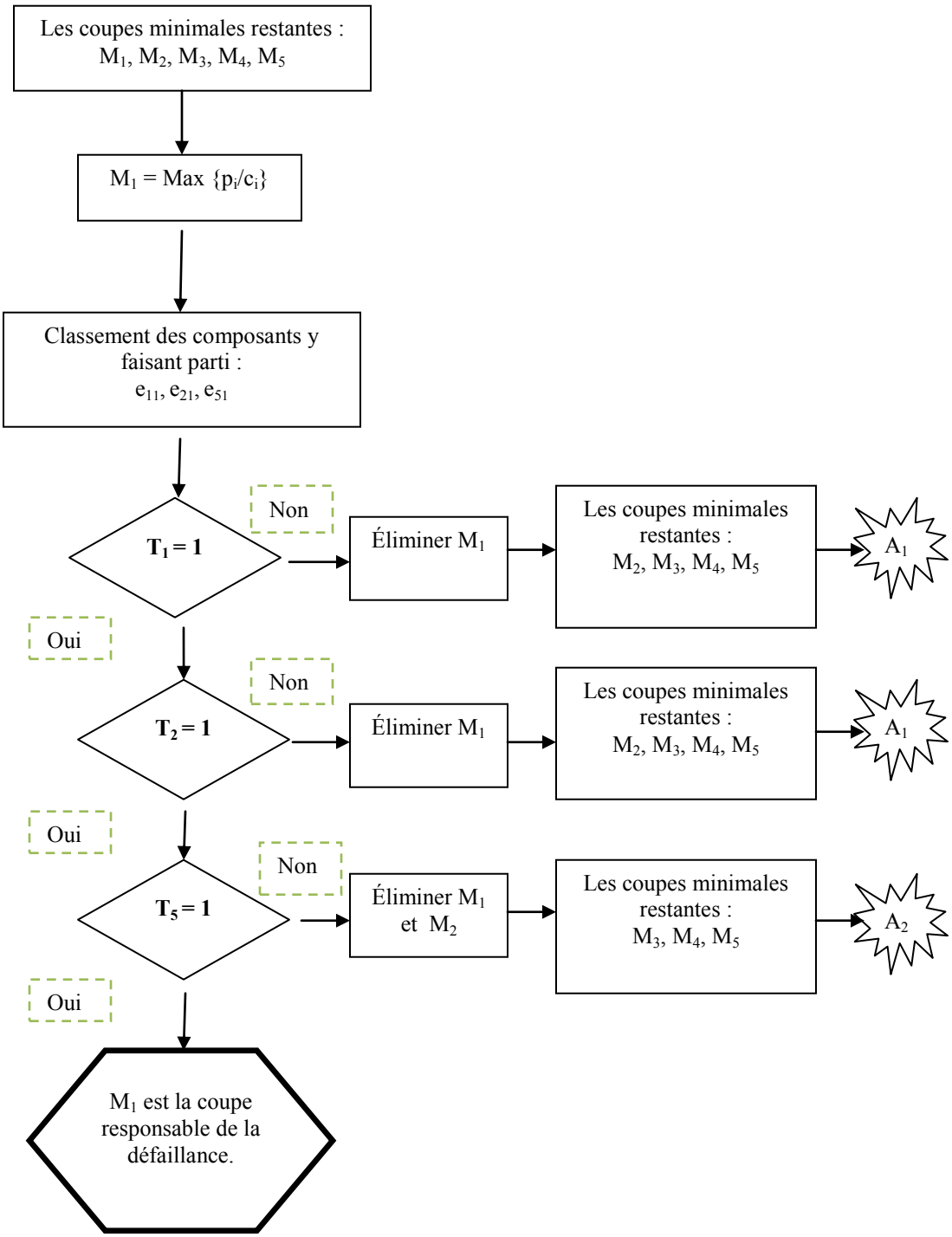

Figure 25. Procédure de tests relative à l'exemple 3

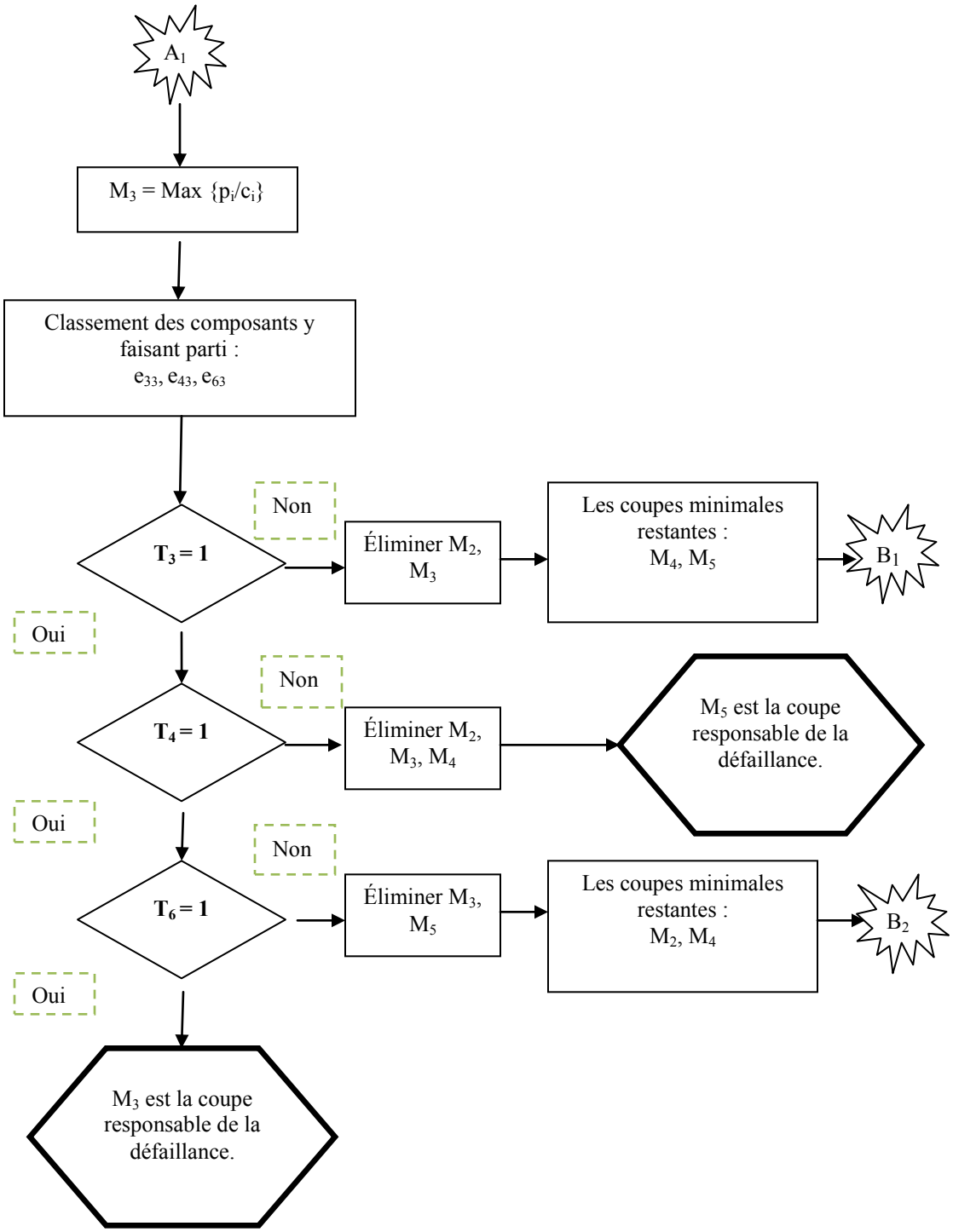

Figure 26. Procédure de tests relative à l'exemple 3 (2)

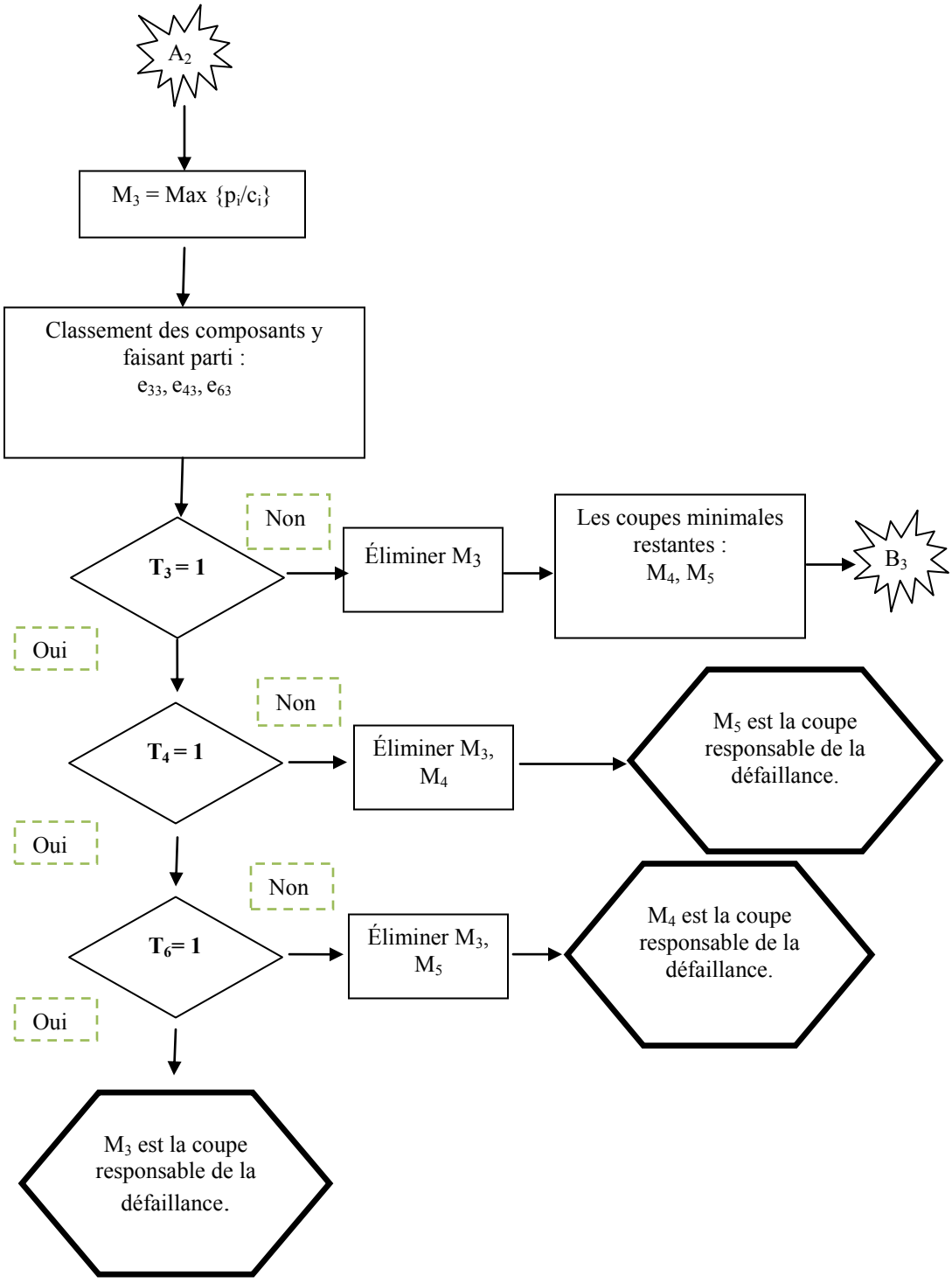

Figure 27. Procédure de tests relative à l'exemple 3 (3)

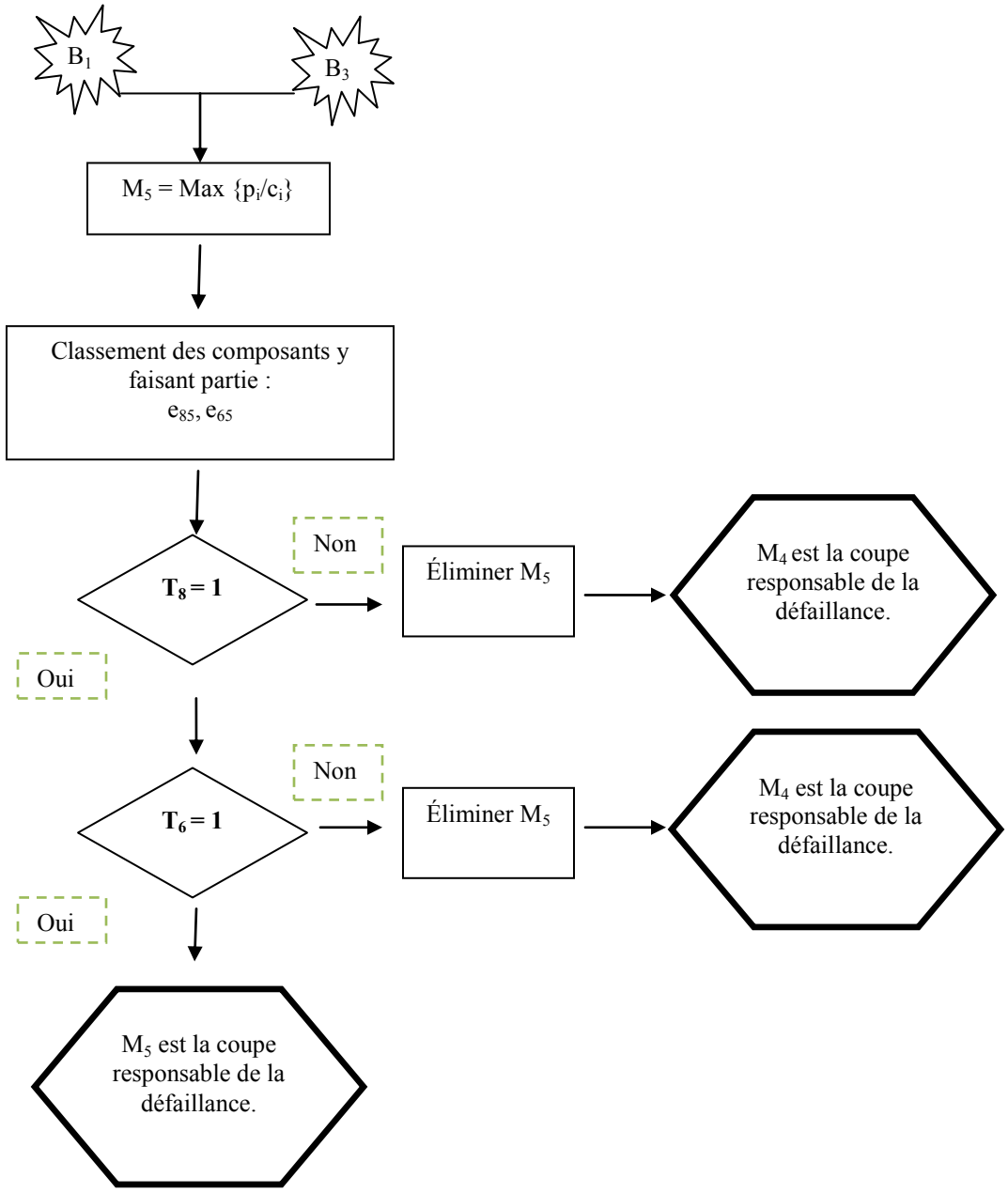

Figure 28. Procédure de tests relative à l'exemple 3 (4)

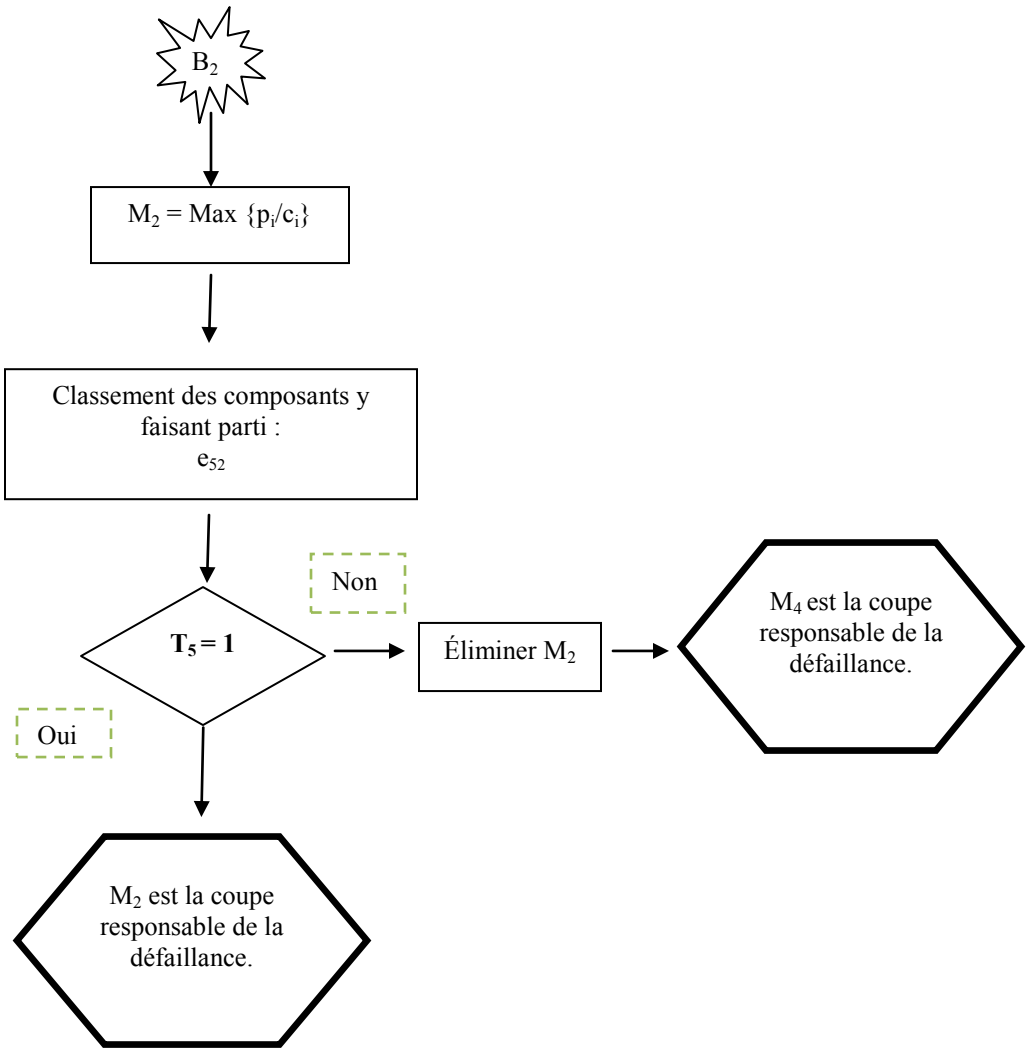

Figure 29. Procédure de tests relative à l'exemple 3 (5)

Chapitre 6 : Conclusion et perspectives

1. Conclusion

Pour un système multicomposant utilisant plusieurs technologies, il est souvent difficile de localiser le (les) composant(s) responsable(s) de la défaillance. Le technicien de maintenance va souvent faire appel à son expérience pour résoudre le problème, mais, dans un cas où la maintenance exige des délais relativement courts pour une efficacité maximale, ces procédés ne sont toujours pas simples à réaliser.

Diverses stratégies de diagnostic ont été définies dans ce travail. Des exemples empruntés à la littérature sont traités pour expliquer chaque stratégie.

Dans la première partie de ce mémoire, on s'est intéressé aux systèmes séries simples où un seul composant peut tomber en panne. L'approche probabiliste a été utilisée pour tester les composants un à la fois.

Dans le cas où un test peut révéler plusieurs états de défaillances, un dictionnaire de tests binaires a été employé. En ce moment, la théorie d'information a été mise en œuvre pour la conversion des tables de décision en arbres de décision. Comme la construction des arbres de décision optimale est un problème NP-complet, les heuristiques ont été exploitées pour remédier au problème d'une façon quasi-optimale. Deux approches ascendantes et descendantes ont été illustrées dans ce travail. Des algorithmes portant sur la programmation dynamique et le gain d'information ont été recommandés. Les séquences de tests formées sont des séquences dynamiques vu que la procédure est choisie au fur et à mesure pendant la construction de l'arbre de décision.

Malgré leurs performances, les modèles utilisés ne produisent pas de bons résultats face à un nombre élevé d'états de défaillances. Pour pallier ce problème, les stratégies de déploiement sont utilisées. Ces stratégies sont souvent combinées aux principes de la théorie d'information et les algorithmes heuristiques pour offrir une solution sous-optimale avec des économies considérables dans les calculs effectués.

Dans le 5^{éme} chapitre de ce mémoire, on s'est focalisé sur les systèmes complexes où plusieurs composants peuvent tomber en pannes simultanément. Le travail est basé sur les résultats du diagramme de fiabilité, les coupes minimales et les arbres de décision.

2. Perspectives

Les coûts de tests

La plupart des travaux sont basés sur les coûts de tests. Ces coûts sont déterminés avant chaque inspection. Pour mieux se rapprocher de la réalité, on peut envisager que ces coûts sont variables en fonction des résultats des tests précédents. En effet, le test d'un composant demande le montage du système. Dans plusieurs cas, le test du composant (i-1) peut diminuer le coût de test du composant (i) vu qu'il est plus accessible.

Le dictionnaire de tests multimodes

Dans le cas d'un système hybride fonctionnant en mode multiple, on peut parler d'un dictionnaire de tests à plusieurs modes. En ce moment, des coûts de transition sont engendrés lors du changement de modes. Plusieurs algorithmes ont été présentés dans divers travaux. D'une façon générale, pour chaque itération, ces algorithmes évaluent la capacité des modes pour choisir celui qui offre la capacité maximale. Après, au sein du mode choisi, on applique le test qui possède le gain d'information le plus élevé. Nous proposons également le recours aux stratégies de déploiement qui sont garanties à améliorer la performance des heuristiques.

Les tests à plusieurs valeurs

Dans les prochains travaux, on pourra se pencher sur les séquences de tests à plusieurs valeurs. En ce moment, chaque test peut avoir différents résultats désignant plusieurs comportements. Un algorithme basé sur une fonction d'évaluation heuristique fera l'objectif recherché.

Surveillance en continu

Pour faciliter le diagnostic, on peut rajouter le contrôle de certains paramètres critiques. Ainsi, une installation de plusieurs capteurs peut être accomplie. Cette installation doit assurer une surveillance en continu pour fournir l'information souhaitée permettant la rapidité et l'efficacité maximale de la phase de diagnostic.

Bibliographie

1. Ait-Kadi D., Gao J. et Portmann M. C., *Minimisation des coûts de détection de pannes à partir des coupes minimales du diagramme de fiabilité*, Troyes, France, 3éme Conférence francophone de Modélisation et Simulation, 2001
2. Barthelemy S., *Introduction aux Réseaux de Neurones,* Economics, Finance & Datamining, 2000.
3. Bhandari S., Simon A. et Siewiorek, P., *Optimal probe selection in diagnostic search*, IEEE Trans. on systems, man, and cybernetics, 1990, vol. 20, nº 5.
4. Canfield R. V., Nachlas J. A., *Comment on: Diagnostic-Strategy Selection for a Series-System*, IEEE Trans. on reability, 1991, vol. 40, nº 2, p 155.
5. Chandrasekaran B. et Milne R., *Special section on Reasoning about structure, behavior and function*, SIGART Newsletter, 1985, nº 93, pp 4-55.
6. Emond J.-c., *les types de règles et leur utilisation dans le domaine du diagnostic technique.*
7. Fellouah R.**,** *Contribution au Diagnostic de Pannes pour les Systèmes différentiellement plats*, Toulouse, France, 2007, pp 15-16.
8. Gao J., *Surveillance et diagnostic de pannes*, thèse de doctorat, département de génie mécanique, université Laval, Québec, CANADA, 1997 (non publié).
9. Garey R. et Graham R. L., *Performance bounds on the splitting algorithm for binary testing*, Acta. Inform., 1974, vol. 3, pp 347–354.
10. Giraud L., *Génération automatique de séquences de tests pour la détection rapide de pannes*, département de génie mécanique, université Laval, Québec, CANADA, 1995.
11. Gluss B., *Optimum policy for detecting a fault in a complex system*, Operation research, 1959, vol. 7, nº 4, pp 468-477.
12. Gluss B. et Firstman S. T., *Optimum search routine for automatic fault location*, Operation research, 1960, vol. 8, nº 4, pp 511-522.

13. Gould E., *Modeling it both ways: Hybrid Diagnostic modeling and its application to hierarchical system designs*, IEEE Autotestcon, 2004.
14. Haykin S., *Neural networks: A comprehensive foundation*, NY, Macmillan, 1994, p 2.
15. Le dépannage des machines, *La méthodologie du diagnostic*, [en ligne], http://www.crta-avignon.com/dossiers/Methode_diagnostic_de_panne.pdf, 2010.
16. Mahdaoui R. et Mouss H.L., *Diagnostic industriel par Neuro-Floue: Application à un système de Production*, Laboratoire d'Automatique et Productique (LAP) Université de Batna: 4th International Conference on Computer Integrated Manufacturing, 2007.
17. Mekroud N. et Moussaoui A., *Approche hybride pour le diagnostic industriel basée sur le RàPc et le Datamining: utilisation de la plate forme JCOLIBRI 2.1*, Sétif, Université Ferhat Abbas, 2009.
18. Mezroua M. A., *Méthodes et techniques de diagnostic,* [en ligne], http://mezrouamedali.centerblog.net/4-Methodes-et-techniques-de-diagnostic, 2010.
19. Milne R., *Strategies for Diagnosis*, IEEE transactions on systems, man, and cybernetics, 1987, vol. SMC-17, nº 3.
20. Nachlas J. A., Loney S. R. et Binney B. A., *Diagnostic-Strategy Selection for Series-System*, IEEE Trans. on reability, 1990, vol. 39, nº 3, pp 272-281.
21. Pattipati K.R. et Alexandridis M. G., *Application of heuristic search and information theory to sequential fault diagnosis,* IEEE Trans., 1990, vol. 20, nº 4.
22. Pattipati Krishna R. et Fang T., *Rollout Strategies for Sequential Fault Diagnosis*, IEEE Trans. on systems, man, and cybernetics - partie A, systems and humans, 2003, vol. 33, nº1.
23. Pattipati Krishna R., Ded S., Dontamsetty M. et Maitra A., *START: system testability analysis and research*, IEEE AES systems magazine, 1991, vol. 6, nº 1, pp 12-21.
24. Pau L., *Diagnostic de panne dans les systèmes-Approche par la reconnaissance des formes*, Toulouse, France, 1975, p 210.
25. Talon A., Boissier D., Peyras L., *Analyse des Modes de Défaillances, de leurs Effets et de leur Criticité*, [en ligne], http://webdav-noauth.unit-c.fr/files/perso/hniandou/cyberrisques2/etage_3_aurelie/co/Module_Etage_3_synthese_49.html, Mars 2009.

26. Varshney P. K., Hartman C. R. P et J. M. de Faria, *Application of information theory to sequential fault diagnosis*, IEEE Trans. on computers, 1974, vol. 31, pp 164-170.
27. Varshney P. K. et Hartman C. R. P, *Sequential fault diagnosis of modular system*, IEEE Trans. on computers, 1984, vol. 33, pp 193-197.
28. Wikipédia, [en ligne], http://www.wikipedia.org/.