N° d'ordre : **2480**

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE SCIENCES PHYSIQUES ET DE L'INGÉNIEUR

Par Philippe THOMAS

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : Automatique, Productique

****

Contribution à l'approche booléenne de la sûreté de fonctionnement : L'atelier logiciel Aralia WorkShop

****

Soutenue le : 6 février 2002

Après avis de :

M. CHATELET Eric, Professeur	Rapporteurs
M. LABEAU Pierre-Etienne, Research Associate FNRS,	

Devant la Commission d'examen formée de :

M. CHATELET Eric, Professeur,	Président
M. DUCAMP François, Industriel IPSN,	Examinateur
M. DUTUIT Yves, Professeur,	Examinateur
M. GRIFFAULT Alain, Maître de Conférences,	Rapporteur
M. HUTINET Tony, Industriel IXI (GFI Consulting),	Examinateur
M. LABEAU Pierre-Etienne, Research Associate FNRS,	Examinateur
M. RAUZY Antoine, Chargé de recherche CNRS,	Examinateur

- 2002 -

RESUME

Cette thèse synthétise les travaux conduits et les résultats obtenus dans le cadre de deux projets de recherche soutenus par des partenaires industriels et décrits ci-après:

- Le projet **Hévéa** visait à traduire les séquences des arbres d'événements (AdE) en formules booléennes. La première phase du projet a abouti à une formalisation du problème et à la réalisation d'un logiciel permettant cette traduction. La deuxième phase a permis de trouver des solutions permettant de diminuer les problèmes d'explosion combinatoire du traitement par diagrammes binaires de décision (BDD : Binary Decision Diagram) des formules booléennes ainsi générées. La troisième phase a validé les approximations généralement utilisées lors des Etudes Probabilistes de Sûreté (EPS) des centrales nucléaires au terme d'une étude systématique portant sur deux EPS industrielles.
- Le projet **Aloès** avait pour but d'étudier des problèmes d'allocation d'exigences s'appuyant sur des modèles booléens d'analyse des risques. Après une recherche bibliographique et des discussions avec les partenaires du projet, il a abouti à la création d'un langage générique de description de problèmes d'allocation. Ce langage, ainsi que le logiciel **Aloès** supportant le langage, sont suffisamment génériques pour être couplé à des moteurs des calculs externes comme **Aralia** (voir ci-dessous) ou **Réséda** (outil traduisant un réseau de fiabilité en formule booléenne). Le logiciel offre alors trois algorithmes d'optimisation (descente rapide, Branch & Bound, Simplex) afin de résoudre les problèmes d'allocation. Des expérimentations sur un ensemble de réseaux de fiabilité ont permis de valider le pouvoir d'expression du langage, mais ont conclu sur la nécessité d'améliorer les algorithmes d'optimisation intégrés.

L'ensemble de ces travaux a débouché sur la création de l'atelier logiciel **Aralia WorkShop** de saisie et de traitement des modèles booléens d'analyse des risques (Arbre de défaillance : AdD, Arbre d'événements : AdE, Bloc diagramme de fiabilité : BDF), s'appuyant pour la partie traitement sur le cœur de calcul **Aralia**. Ce dernier se base sur un codage des formules booléennes en BDD.

Mots-Clés :
Atelier SdF, Arbre d'événements, Arbre de défaillance, Réseau de fiabilité, Défaillance de Cause Commune, Loi test périodique, BDD, Diagramme Binaire de Décision, Formule booléenne, Réordonnancement, Réécriture, Allocation d'exigence, Branch & Bound, Optimisation

ABSTRACT

This thesis presents the results obtained by the author during his participation to two research programs supported by industrial partners:

- The **Hévéa** projet
  The main objective of this project was to assess event-tree sequences by means of Boolean techniques. During the first stage of this program, the author formalized the problem. He designed the Hévéa software which is dedicated to event-tree assessment. In the second stage, the author introduced several original means to reduce the exponential blow-up of the BDD-based processing of Boolean formulae. During the third stage, he validated by means of a systematic study of two industrial PSA (Probabilistic Safety Analysis) the approximations commonly used in the nuclear framework.
- The **Aloes** project
  The aim of this project was the study of so-called requirement allocation problems. This project began by an important bibliographic work and ended by the creation of a language suitable for allocation problems. This language, and related software Aloès are generic enough to be connected to external assessment softwares such as **Aralia** (which encodes boolean formulae by using BDD) and **Réséda** (which transform a reliability network into boolean formulae). Three optimisation algorithms (steepest descent, branch and bound, simplex) are implemented in the **Aloès** Software to solve allocation problems. Several experiments on reliability networks validated language. These experiments showed that some of previous embedded optimisation algorithms must be improved.

The final result of all these studies is the software set **Aralia WorkShop** which is devoted to the assessment of Boolean risk analysis models (Faut-tree, Event-tree, Reliability block-diagrams and networks) thanks to the embedded computing software **Aralia**.

Keywords :
RAMS workshop, Event tree, Fault tree, Reliability network, Common cause failure, Periodic test law, BDD, Binary Decision Diagram, Boolean formula, Reordering, Rewriting, Requirement allocation, Branch & Bound, Optimisation

REMERCIEMENTS

Aux rapporteurs de ce document, Pierre-Etienne Labeau et Eric Châtelet pour leurs lectures soignées, leurs remarques pertinentes et les réflexions qui en ont découlé.

Aux partenaires industriels des différents projets auxquels j'ai eu l'occasion de participer. Je remercie en particulier François Ducamp (IPSN : Institut de Protection et de Sûreté Nucléaire) pour son soutien actif et global à Aralia WorkShop et Jean-Pierre Signoret (ELF Exploration-Production) pour ces réflexions souvent à propos.

A la société IXI (depuis peu GFI Consulting). A ses dirigeants qui ont accepté qu'une ressource productive soit détachée pendant 3 années pour des activités de recherche. A Tony Hutinet, sans lui la synergie entre les différents projets et outils autour de la sûreté de fonctionnement n'auraient jamais aussi bien fonctionné.

A André Arnold et à Richard Castanet, les deux directeurs du LaBRI (Laboratoire Bordelais de Recherche en Informatique) qui se sont succéder durant ma thèse, pour m'avoir accueillit au sein de ce laboratoire. Aux membres de l'équipe MVTSi (Modélisation, Vérification et Test des Systèmes Informatiques) et en particulier à Alain Griffault pour leur gentillesse et leur aide ponctuelle. Aux membres de la défunte CVT (Cellule de Valorisation et de Transfert). Je remercie naturellement Gérald Point pour tous les moments partagés ses 4 dernières années.

A Yves Dutuit pour m'avoir fait découvrir le monde de la recherche, pour son écoute attentive, sa bonne humeur et pour tout le temps qu'il a consacré à remettre en forme mes idées quelque fois brouillonnes.

A Antoine Rauzy pour toutes les discussions formelles, informelles et/ou philosophique et aussi pour ses relances perpétuelles lors de la rédaction de ce document.

A toutes les autres personnes que j'ai rencontré durant ces quelques années et qui m'ont apporté d'une manière ou d'une autre l'énergie nécessaire pour mener ce travail à terme.

A mes amis et ma famille avec une mention spéciale à Isa, Robin et Jolan (pour m'avoir supporté ces derniers mois !).

TABLE DES MATIERES

1. INTRODUCTION

Les travaux présentés dans ce mémoire ont été réalisés dans le cadre d'une convention CIFRE (Convention Industrielle de Formation à la Recherche) entre la société IXI (Ingénierie Concourante et Systèmes d'Information) et le Ministère de l'Education Nationale, de la Recherche et de la Technologie. Ils se sont déroulés au sein de l'équipe MVTsi (Modélisation, Vérification et Test des Systèmes Informatisés) du LaBRI (Laboratoire Bordelais de Recherche en Informatique) et en collaboration avec l'équipe ADS (Analyse des Dysfonctionnements des Systèmes) du LAP (Laboratoire d'Automatique et de Productique) de Bordeaux.
La société IXI a, ces dernières années, mis en place une stratégie de diffusion et de développement d'outils dédiés à l'analyse des risques. Dans un premier temps distributeur du logiciel Aralia, IXI a souhaité s'investir d'une manière durable, comme partenaire à part entière du LaBRI, dans des projets pouvant réunir d'autres partenaires industriels et débouchant sur la réalisation d'outils informatiques commercialisables. C'est ainsi qu'IXI a financé deux thèses CIFRE au LaBRI : celle de Gérald Point dans le cadre du projet AltaRica, et la mienne (toutes deux encadrées par Antoine Rauzy et, pour la seconde avec le concours d'Yves Dutuit). Ces différents projets ont effectivement abouti à la réalisation d'un pôle outils qui offre une gamme de logiciels d'analyse des risques commercialisés, distribués, maintenus et/ou développés par IXI.
Durant ma thèse, j'ai travaillé principalement sur un atelier de sûreté de fonctionnement spécialisé dans les modèles booléens d'analyse des risques : Aralia WorkShop. Ces modèles sont encore actuellement les plus utilisés dans le monde industriel. Le développement de l'atelier Aralia Workshop, autour du cœur de calcul Aralia, était au centre de la stratégie de IXI et a été soutenu par l'ensemble des partenaires industriels déjà impliqués dans d'autres projets.
Le prochain paragraphe de ce chapitre présente la pierre angulaire de cette thèse : le logiciel Aralia. Les deux suivants sont consacrés à deux projets directement issus de travaux centrés sur ce logiciel: les projets Hévéa et Aloès. Le premier a pour but d'étudier la traduction des arbres d'événements en formules booléennes et de résoudre les problèmes d'explosion combinatoire pouvant en résulter. Le second est consacré à la résolution de problèmes d'allocation d'exigences (au sens large) et plus particulièrement d'allocation d'indisponibilité.
L'avant-dernier paragraphe de ce chapitre présente les différents logiciels du pôle outils. Mes travaux sur les outils, autres que ceux du pôle booléen, y sont également décrits succinctement. Le dernier paragraphe de cette introduction est consacré à une brève présentation de l'atelier Aralia WorkShop.

1.1 LE PROJET ARALIA

Le projet Aralia est le fruit d'une collaboration entre deux laboratoires de l'Université Bordeaux I (à l'époque le LADS et le LaBRI) et plusieurs sociétés (par ordre alphabétique: CEA/IPSN, Dassault, EDF, Elf-EP, SGN, Renault, Schneider Electric et Technicatome). Cette collaboration a vu le jour en 1993 lors d'une réunion du Groupe de Recherche Méthodologique de l'Institut de Sûreté de Fonctionnement (ISdF) et s'est concrétisée par la réalisation d'un groupe de travail connu depuis sous le nom de groupe Aralia. L'objectif de ces membres était d'étudier ce que les BDD (Binary Decision Diagram ou Diagrammes Binaires de Décision) pouvaient apporter aux modèles booléens d'analyse des risques.

Cette collaboration a réussi au delà de toute espérance puisqu'elle a donné lieu à de nombreuses publications scientifiques [Arl94, Arl95, ...] et a débouché sur la réalisation du logiciel Aralia. Aralia a été développé (et est toujours développé) par Antoine Rauzy, en collaboration avec Yves Dutuit, pour les aspects mathématiques et fiabilistes. Aralia est un cœur de calcul destiné à être utilisé par différents outils. Pour cette raison, son interface utilisateur consiste en un interprète de commandes (du type shell Unix). Mais, comme il n'est pas envisageable qu'un ingénieur fiabiliste utilise une interface aussi rudimentaire, des interfaces de saisie conviviales comme GRIF ou SimTree ont été couplées à Aralia dès le début du projet. Ces dernières ont ainsi permis aux partenaires industriels du groupe d'utiliser Aralia et de valider les résultats qu'il fournit. L'interface SimTree a été réalisée par Jean-Luc Cadiot appartenant alors au LADS, laboratoire dirigé par Yves Dutuit. Par la suite ces logiciels (Aralia/SimTree et Aralia/GRIF) ont été industrialisés et distribués par la société IXI.

Aralia/SimTree s'est progressivement imposé sur le marché français des logiciels de saisie et de traitement d'arbres de défaillance.

A la fin de l'année 1997, un certain nombre de partenaires du groupe souhaitait étudier des problèmes connexes. Deux nouveaux partenariats se sont donc mis en place dans ce but. Ils avaient pour objet :

- la traduction des arbres d'événements en formules booléennes et l'étude des problèmes pouvant en résulter,
- l'allocation d'indisponibilité s'appuyant sur des modèles booléens d'analyse des risques.

Ces deux sujets constituent précisément le cœur de ma thèse.

1.2 PROJET HEVEA

Depuis une vingtaine d'années et, plus précisement, depuis la parution du rapport WASH-1400 [RSS75], l'utilisation conjointe de la technique des arbres d'événements et de celle des arbres de défaillance (ET/FT) s'est progressivement imposée dans l'ingénierie nucléaire. Un arbre d'événements code un ensemble de séquences d'événements. Chaque séquence commence par un événement initiateur, comprend un certain nombre d'événements génériques et conduit le système modélisé dans un état caractéristique (panne récupérée, panne non récupérée, …). Les événements génériques sont souvent définis par des arbres de défaillance. La quantification des arbres d'événements pose deux problèmes mathématiques difficiles à résoudre, liés à l'interdépendance des événements génériques. Il existe en effet deux types de dépendances entre ces événements :

- Des dépendances fonctionnelles : les arbres de défaillance définissant deux événements génériques peuvent partager des événements de base. Si tel est le cas, le calcul de la probabilité d'occurrence d'une séquence les contenant tous les deux doit tenir compte de ce type de dépendance.
- Des dépendances temporelles : une séquence dans l'arbre d'événements décrit une trajectoire du système modélisé. L'ordre dans lequel les événements surviennent peut être important. Par exemple, l'instant de démarrage d'un composant en redondance passive dépend de l'instant de détection de la panne du composant principal. Il s'ensuit que la probabilité d'occurrence d'une séquence ne peut pas toujours être calculée en faisant le produit des probabilités d'occurrence des événements la composant.

Avec Antoine Rauzy, j'ai proposé une traduction des arbres d'événements en équations booléennes. Cette traduction permet de résoudre le premier des deux problèmes ci-dessus, mais ne résout pas la prise en compte des dépendances temporelles entre événements génériques.

Ce travail s'est effectué dans le cadre d'un partenariat entre différents organismes (le LaBRI d'une part, CEA-IPSN, Elf-EP, Technicatome et IXI d'autre part). Ce partenariat s'est fixé comme objectif la réalisation d'un logiciel de traitement d'arbres d'événements associé à Aralia/SimTree.

Le logiciel que j'ai réalisé comprend en fait deux parties : d'une part, le cœur de calcul Hévéa qui compile les arbres d'événements en équations booléennes destinées à être traitées par Aralia ; d'autre part, une interface graphique conviviale permettant la saisie des arbres d'événements et la connexion avec le cœur de calcul.

Le projet Hévéa a comporté trois phases distinctes, présentées respectivement dans les chapitres 5, 6 et 7. Elles sont résumées ci-après :

1. La première phase a eu comme objectif la formalisation mathématique des arbres d'événements et leur traduction en formules booléennes [TR99], ainsi que la réalisation du logiciel Hévéa. Les études réalisées par les partenaires industriels du projet ont mis en évidence la difficulté de traiter les formules booléennes générées en raison de leurs tailles.
2. La deuxième phase a été consacrée à l'étude des moyens à mettre en œuvre pour traiter ces problèmes d'explosion combinatoire [DMRST00], [DPRT00]. Je me suis consacré à la mise au point de différentes méthodes de réécritures des formules booléennes permettant de simplifier leur traitement. Antoine Rauzy s'est dans le même temps chargé de la réalisation d'un algorithme "à la MOCUS" (Cf. chapitre 2.5) au sein d'Aralia, ainsi que de la mise au point de stratégies spécifiques d'utilisation des diagrammes binaires de décision. A l'issue de ces travaux, tous les arbres d'événements fournis par les partenaires industriels ont pu être quantifiés de manière approchée et exacte.
3. La dernière phase du projet a eu pour objet l'étude des différentes approximations utilisées dans les logiciels classiques de quantification des arbres d'événements booléens. Ces approximations ont été simulées à l'aide d'algorithmes disponibles au sein d'Aralia. Deux études probabilistes de sûreté de taille

conséquente ont servi de références pour comparer les résultats exacts et ceux issus de différentes approximations.

1.3 PROJET ALOES

Le but d'un outil ou d'une méthode d'allocation de fiabilité est d'aider à la conception ou à l'amélioration de systèmes de plus en plus complexes dans un environnement économique de plus en plus concurrentiel. L'allocation de fiabilité vise à trouver un juste équilibre entre différentes grandeurs caractéristiques d'un système donné, tel que son coût ou sa fiabilité. Il existe fondamentalement deux approches d'allocation.
La première est une démarche descendante. Elle consiste à attribuer des objectifs à chaque composant d'un système, en fonction de son type et de paramètres de pondération, de telle façon que l'objectif global au niveau du système soit atteint.
La seconde, plus générale, est une démarche ascendante. Elle consiste à chercher une solution répondant à des critères d'optimalité en jouant sur des variables de décision qui représentent les choix d'organisation, de qualité des composants, de politique de maintenance, d'architecture système,... que l'équipe de conception souhaite étudier. Les paramètres fiabilistes des composants, leurs coûts élémentaires,... sont directement définis en fonction de ces variables de décision.
Je me suis intéressé principalement à la seconde approche dite de l'allocation par optimisation.
Les méthodes d'optimisation proposées dans la littérature sont nombreuses et présentent toutes des singularités. Elles se différencient sur de nombreux points tels que :

- le type de systèmes pris en compte (série - parallèle, quelconque,...),
- les modèles fiabilistes considérés (arbres de défaillance, graphes de Markov,…),
- la (ou les) grandeur(s) à optimiser,
- la prise en compte ou la non prise en compte de contraintes (de poids, de coût, de fiabilité,…),
- la nature des variables de décision (discrètes ou continues),
- le type de fonction permettant de définir les choix,
- les algorithmes d'optimisation, ...

La difficulté, pour une équipe de conception, est de trouver, parmi toutes les méthodes d'optimisation disponibles, celle qui sera plus adaptée au type de problème qu'elle a à résoudre.
Ne souhaitant pas imposer aux utilisateurs une formalisation particulière du problème d'allocation, nous avons proposé de le décrire à l'aide d'un langage : le langage Aloès. La seule limite que nous nous sommes imposée concerne le type de modèles fiabilistes utilisés. Ce sont les modèles booléens d'analyse des risques, c'est-à-dire les arbres de défaillance et les réseaux de fiabilité.
Ce travail s'inscrit dans le cadre d'un partenariat entre différents organismes (le LaBRI d'une part, Alstom, CEA-IPSN, EDF, Elf-EP, Technicatome et IXI d'autre part). Ce partenariat s'est fixé comme objectif la réalisation d'un logiciel d'optimisation associé aux logiciels de traitement des formules booléennes Aralia et de traitement des réseaux de fiabilité Réséda. Certains partenaires (Technicatome et EDF) avaient déjà une certaine expérience sur l'allocation d'indisponibilité. Chacun d'eux avait en effet développé et utilisé un outil d'allocation (respectivement Sherloc [MRT97, DMRT98a, DMRT98b, DMRT98c] et ARPO [BB97a, BB97b]).
Le logiciel que j'ai réalisé est un interprète de commandes prenant en entrée des descriptions de problèmes d'allocation écrites dans le langage Aloès. Il offre trois algorithmes de résolution qui se différencient par leur caractère exhaustif ou stochastique ainsi que par la nature des variables de décision qu'ils peuvent prendre en considération (variables discrètes ou continues).
Le langage Aloès et les algorithmes considérés sont présentés au chapitre 10. Ils ont été validés sur de nombreux exemples de la littérature. Le chapitre 11 est consacré à une étude d'allocation sur un ensemble de réseaux de fiabilité. Cette étude aborde trois problèmes différents : l'allocation de redondance, l'optimisation de maintenance et l'allocation en conception préliminaire.
En marge de cette étude, je me suis intéressé à un aspect plutôt méthodologique qui concerne la notion de coût d'un système et à sa définition en fonction du coût des composants. Le chapitre 9 présente une étude des différentes fonctions de coût proposées dans la littérature.

1.4 LE POLE OUTILS

Comme nous l'avons mentionné, la société IXI a mis en place, à l'initiative de Tony Hutinet, une stratégie de développement et de diffusion d'outils d'analyse des risques. Les arguments en faveur du développement du pôle outils sont nombreux :

- La commercialisation et/ou le développement de logiciels peuvent être rentables.
- Ces actions favorisent l'acquisition de nouvelles connaissances en étroite relation avec la recherche.
- L'usage de ces outils en interne permet la réalisation d'étude de sûreté de fonctionnement à moindre coût.
- C'est un vecteur de dissémination important permettant de soutenir des actions d'études et de conseils vis-à-vis des clients d'IXI.
- Le pôle outils renforce l'image de marque de l'entreprise.

La Figure 1.1 recense l'ensemble des logiciels inclus d'une manière ou d'une autre dans le pôle outils de IXI. Les cœurs de calcul, généralement des logiciels fonctionnant en ligne de commandes, sont représentés par des ovales et les interfaces de saisie conviviales par des parallélogrammes. Les outils en trait gras sur fond blanc sont ceux qui existaient déjà en 1997, avant le début de cette thèse. Ceux sur fond gris ont été réalisés depuis. Le schéma montre que de nombreux outils s'appuient sur le logiciel Aralia.

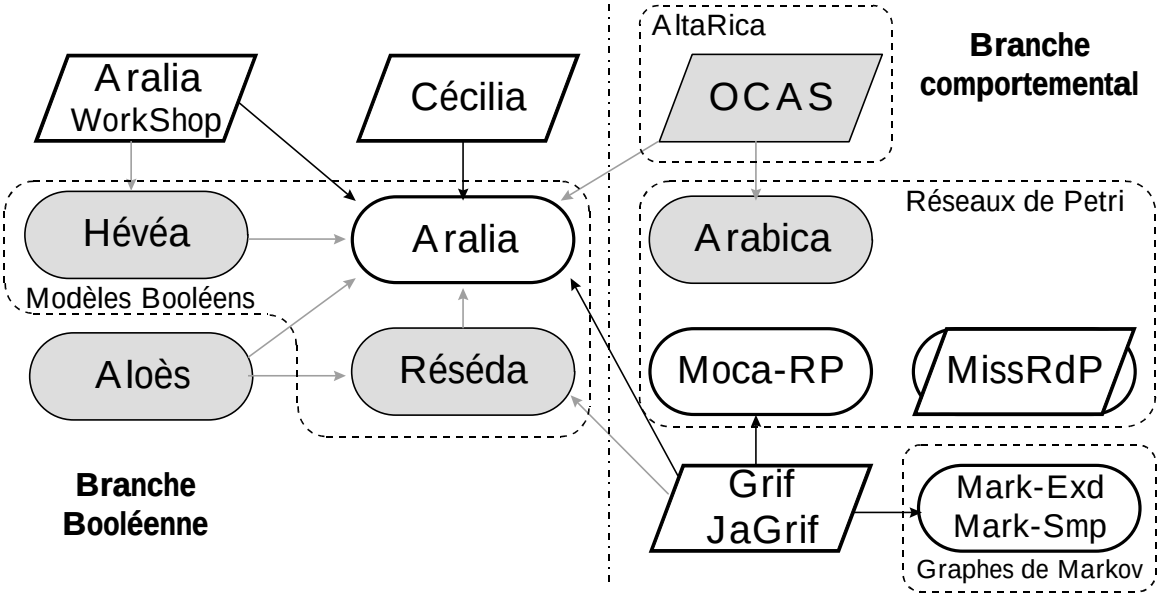


Figure 1.1 : Ensemble des produits du pôle outils

Description des différents cœurs de calcul:

- Aralia a été décrit dans le premier paragraphe du présent chapitre. Il est la pierre angulaire de nombreux outils.
- Réséda (1998, 2000, LaBRI, Elf-EP) est un compilateur de réseaux de fiabilité (Cf. Chapitre 3.1) vers les formules booléennes, qui, à partir d'un réseau, des nœuds source et cible et de directives de compilation, génère un fichier au format Aralia. Ce compilateur a été réalisé par Antoine Rauzy pour Elf-Aquitaine.
- Hévéa (1998, groupe Hévéa : CEA-IPSN, Elf-EP, IXI, LaBRI, Technicatome) est un traducteur d'arbres d'événements couplé à des arbres de défaillance vers des formules booléennes. La partie II de cette thèse lui est entièrement consacrée.

- Aloès (2000, groupe Aloès : Alstom, CEA-IPSN, EDF, Elf-EP, IXI, LaBRI, Technicatome) est un outil d'allocation d'exigences orienté Sûreté de Fonctionnement. Il est présenté en détail dans la partie III.
- Moca-RP (1981, Elf-EP) effectue des simulations de Monte Carlo sur la base de réseaux de Petri stochastiques interprétés. Les extensions conventionnelles des réseaux de Petri (messages, arcs inhibiteurs, arcs pondérés, ...), de nombreuses lois de probabilité (exponentielle, Weibull, Erlang généralisée, Dirac, Périodique, ...) et des règles spécifiques permettant de traiter les défaillances à la sollicitation font de Moca-RP un outil polyvalent d'analyse de la sûreté de fonctionnement des systèmes. La dernière version a été développée par Gérald Point et a servi de base pour le projet Arabica.
- Arabica (1998, groupe Arabica : CEA-IPSN, Dassault Aviation, Elf EP, IXI, LaBRI, Schneider, Technicatome) a eu pour ambition d'étudier les réseaux de Petri de manière qualitative via la détermination des séquences partant de l'état initial et aboutissant à un état redouté du système étudié.
- MissRdP (1988, IXI en collaboration avec le LAAS) permet de manipuler des réseaux de Petri stochastiques colorés. Le modèle de MissRdP permet de travailler sur des variables (réelles, entières ou booléennes), tant sur les gardes (ou prédicats) des transitions que sur les actions résultantes du tir d'une transition. Cette richesse de modélisation se paye par des temps de calculs non négligeables. Le cœur de calcul de MissRdP n'est pas disponible en dehors de l'interface.
- Mark-Exd/Mark-Smp (1985, Elf-EP) permet de traiter quantitativement des graphes de Markov à partir d'un algorithme appelé « exponentiation directe ». Mark-Smp est une extension au traitement des systèmes réparables multi-phases.

Présentation des différentes interfaces de saisie graphique :

- Aralia WorkShop (1995, groupe Aralia, Visual C^{++}), anciennement connu sous le nom de SimTree, offre l'interopérabilité des modèles booléens d'analyse des risques. Développé en Visual C^{++}, cette interface séduit par son aspect intégré à Windows (presse-papier, gestion Ole, exportation vers Excel, ...). Son couplage avec Aralia est important, tant dans la définition des calculs à réaliser que pour la récupération et l'affichage des résultats fournis par le cœur de calcul. Cette interface est décrite dans le chapitre 4.
- GRIF-JaGRIF (1992, Elf-EP, Java), propose une interface similaire pour saisir différents modèles d'analyse des risques que sont les arbres de défaillance, les réseaux de fiabilité, les graphes de Markov et les réseaux de Petri stochastiques. Chaque module est couplé avec un logiciel permettant de traiter le modèle considéré.
- Cécilia (1995, Dassault Aviation, Java) permet lui aussi de saisir des arbres de défaillance. Il se différencie de Aralia WorkShop et de GRIF par l'utilisation d'une base de données Oracle pour le stockage des différents modèles. Cette base de données permet un accès concourant (par plusieurs personnes). Cécilia est donc recommandé pour les projets de taille importante faisant intervenir plusieurs équipes travaillant simultanément.

Présentation de AltaRica et de O.C.A.S.

En décembre 1996, les sociétés Elf-EP, IXI et l'équipe MVTsi du LaBRI ont initié un projet qui avait pour ambition d'unifier à terme la sûreté de fonctionnement et la vérification de modèle autour d'un seul outil **:** l'atelier d'analyse système AltaRica. La thèse [Poi00] de Gérald Point, née d'une première collaboration de recherche entre la société IXI et le LaBRI, définit le cadre formel utilisé pour le développement des outils intégrés dans cet atelier. Les deux premières phases du projet AltaRica ont abouti à la création d'un langage dont la sémantique est parfaitement définie et à la réalisation de nombreux outils **:** compilateurs vers les formules booléennes, générateur des séquences accidentelles (scénarii de panne), compilateurs vers des vérificateurs de modèle (model-checker) tels que Mec et Toupie, ...
O.C.A.S. (Outil de Conception et d'Analyse Système) a été développé par la société Dassault Aviation pour réaliser des analyses de sûreté de fonctionnement lors de la conception de ses avions. Il s'appuie sur un sous-langage de AltaRica.

En dehors de mes travaux sur la branche booléenne du pôle outil (Aralia WorkShop, Hévéa et Aloès), j'ai également contribué au développement de la branche comportementale. J'ai participé à l'élaboration du

langage AltaRica ([PTLSR98], [SLPTGR98]) et me suis intéressé aux aspects méthodologiques. J'ai ainsi modélisé de nombreux cas-tests. Ce travail a abouti à l'élaboration d'un guide méthodologique [AT97] donnant des recommandations sur les modifications à apporter au langage et sur les fonctionnalités à prendre en compte dans une interface graphique.
Par ailleurs, mes compétences sur les réseaux de Petri stochastiques et ma maîtrise du logiciel Moca-RP ont été mises à profit dans la modélisation de systèmes [DCST97], dans l'étude de systèmes dynamiques [DTRS96], [DRST97] et dans l'étude qualitative de systèmes reconfigurables [PCTD97] qui a été une des bases de réflexion du projet Arabica.

1.5 LE PROJET ARALIA WORKSHOP

Il est important de rappeler que les travaux présentés dans ce document se sont déroulés dans le cadre d'une convention CIFRE. Dans ce contexte, l'industrialisation des produits avait autant d'importance que les thématiques de recherche. Le but de IXI était, *in fine*, de disposer d'outils diffusables. Il n'était donc pas concevable de se limiter à des prototypes sans interface graphique.
J'ai donc consacré beaucoup de temps et d'énergie à la réalisation d'une interface de saisie graphique conviviale **:** Aralia WorkShop. Cet atelier propose actuellement trois modules permettant la saisie d'arbres de défaillance, d'arbres d'événements et de diagrammes de fiabilité. Chacun de ces modules peut être utilisé individuellement. Tous génèrent une formule booléenne qui est ensuite traitée par Aralia.
Leur couplage à Aralia est fort, puisque la majorité des algorithmes présents dans Aralia sont disponibles et paramétrables au sein de Aralia WorkShop, à l'aide de fenêtres de dialogue. Les résultats générés par Aralia sont interprétés par Aralia WorkShop. Ils sont affichés dans des fenêtres de dialogue ou directement au sein des modèles.
La plupart des notions présentes dans Aralia sont également disponibles à partir de Aralia WorkShop. Parmi ces notions citons les constantes booléennes (vrai et faux), les lois de probabilités associées aux événements de base, les paramètres nommés associés aux lois de probabilité et les attributs servant de drapeau afin de sélectionner un ensemble d'événements de base.
Un module permettant le traitement des défaillances de cause commune a aussi été mis en œuvre. Il s'appuie sur la démarche généralement employée dans le milieu nucléaire qui consiste à ne considérer que les défaillances de cause commune entre composants strictement identiques.

La première partie de cette thèse sert de rappel et d'introduction étendue. Elle se clôt au chapitre 4 par une présentation plus détaillée de l'atelier Aralia WorkShop.

1.6 BIBLIOGRAPHIE

[Arl94] Groupe Aralia. *Arbres de Défaillances et diagrammes binaires de décision*. Actes du Premier congrès interdisciplinaire sur la Qualité et la Sûreté de Fonctionnement, pp 47-56. Université Technologique de Compiègne, 1994.

[Arl95] Groupe Aralia. *Computation of Prime Implicants of a Fault Tree within Aralia*. In Proceedings of the European Safety and Reliability Association Conference, ESREL'95, pp 190-202, Bournemouth – England, June 1995.

[AT97] H. Antoniol et P. Thomas. *Projet AltaRica (Analyse Comportementale et Validation) - Méthodologie de Modélisation*. Rapport interne du projet AltaRica. Réf. IXI/BDX/TLS0108/HA/MET.MOD, 1997.

[BB97a] M. Bouissou et E. Bourgade. *Evaluation et allocation d'indisponibilité à la conception des tranches nucléaires : Méthodes et Outils.* Revue Française de Mécanique n°1997-2, pp105-111, 1997

[BB97b] M. Bouissou and E. Bourgade. *Unavailability Evaluation and Allocation at the Design Stage for Electric Power Plant : Methods and Tools.* Proceedings of Annual Reliability and Maintenability Symposium IEEE, pp91-99, 1997

[DCST97] Y. Dutuit, E. Châtelet, J.P. Signoret et P. Thomas. *Dependability modelling and evaluation by using stochastic Petri nets : application to two test-cases*. Reliability Engineering and System Safety, vol. 55, n° 2, pp. 117-124, 1997.

[DMRST00] F. Ducamp, F. Meunier, A. Rauzy, J.P. Signoret et P. Thomas. *Traitements d'arbres d'événements : Problèmes et Solutions.* Actes du 12e Colloque international de fiabilité et de maintenabilité - λμ12, pp. 269-274, 2000.

[DMRT98a] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Sherloc : a Desk Calculator for Availability Allocation in Fault Trees*. In Proceedings of the European Safety and Reliability Association Conference, ESREL'98.

[DMRT98b] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Conception Optimization Based on the Definition of Availability Allocation Taking into Account Cost Constraint.* In Proceedings of the International Conference of Probabilistic Safety Assessment and Management, PSAM'4, volume 2, pages 1591-1596, New-York, 1998. Springer Verlag.

[DMRT98c] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Sherloc : outil d'allocation d'indisponibilité pour les arbres de défaillance*. Actes du 11ième colloque national de Fiabilité et Maintenabilité, λμ11, pp. 278-283, 1998.

[DPRT00] F. Ducamp, S. Planchon, A. Rauzy et P. Thomas. *Handling very large event trees by means of Binary Decision Diagrams*. In Proceedings of the International Conference on Probabilistic Safety Assessment and Management, PSAM'5, Vol. 3, pp. 1447-1452, 2000

[DRST97] Y. Dutuit, A. Rauzy, J.P. Signoret et P. Thomas. *Analyse qualitative et quantitative de la fiabilité d'un système dynamique.* Actes du 2ème Colloque Pluridisciplinaire Qualité et Sûreté de Fonctionnement, pp. 243-250, 1997.

[DTRS96] Y. Dutuit, P. Thomas, A. Rauzy et J.P. Signoret. *Modélisation d'un système dynamique et évaluation de sa sûreté de fonctionnement par réseaux de Petri stochastiques.* In Proceedings of 10e Colloque international de fiabilité et de maintenabilité - λμ10, T2, pp. 648-659, 1996.

[MRT97] F. Meunier, A. Rauzy, and P. Thomas. *Utilisation de la méthode du gradient descendant pour l'allocation de fiabilité*. Actes de la 3ième Conférence Nationale sur la Résolution Pratique de Problèmes NP-Complets, pages 109-112, 1997.

[PCTD97] S. Pasquet; E. Châtelet, P. Thomas and Y. Dutuit. *Analysis of a sequential non-coherent and looped system with two approaches : Petri Nets and Neural Networks*. In proceedings of International Conference on Safety and Reliability, ESREL'97, pp. 2257-2264, 1997.

[Poi00] G. Point. *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement.* Thèse de l'université de Bordeaux 1. 2000

[PTLSR98] G. Point, P. Thomas, S. Lajeunesse, J.P. Signoret et A. Rauzy. *Le langage AltaRica*. Actes du 11e Colloque international de fiabilité et de maintenabilité - λμ11, pp. 119-125, 1998.

[SLPTGR98] J.P. Signoret, S. Lajeunesse, G. Point, P. Thomas, A. Griffault and A. Rauzy. *The Altarica language.* In proceedings of International Conference on Safety and Reliability, ESREL'98, 1998

[TR99] P. Thomas et A. Rauzy. *Hévéa un gestionnaire d'arbres d'événements couplé à Aralia*. Actes du 3ème Congrès internationnal Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita 99, pp. 463-473, 1999.

Partie I :
Préliminaires

2. PRELIMINAIRES MATHEMATIQUES ET ALGORITHMIQUES

Ce chapitre présente les concepts mathématiques et les outils algorithmiques qui seront utilisés dans le reste du document.
La plupart de ces concepts sont des notions classiques de l'algèbre de Boole et du calcul des probabilités. Ils ne sont donc en rien originaux puisqu'ils sont décrits dans toutes les monographies sur la sûreté de fonctionnement (voir, par exemple, [PG80,Vil88,Lim91,AM93,Coc97]). Un bref rappel permet toutefois de préciser les notations employées par la suite.
La notion de « coupe minimale » est en revanche présentée en détail. Cette notion, qui est pourtant au cœur des méthodes booléennes d'analyse du risque, n'avait pas, jusqu'à une date récente, été formellement définie. La présentation qui en est faite dans ce chapitre suit la référence [Rau01a] (qui a été reprise dans le livre [BC01]).
Ce chapitre comprend aussi une brève introduction aux outils algorithmiques utilisés dans le cadre de cette étude : L'algorithme MOCUS [FV72] et les diagrammes binaires de décision [Bry86,BRB90]. On peut faire la même remarque pour MOCUS que pour les coupes minimales : bien que très souvent employé, cet algorithme n'est vraiment détaillé dans aucun article ou livre de référence sur le sujet. La présentation qui en est faite dans ce chapitre suit celle de l'article [Rau02a]. Par contre, la technique des diagrammes binaires de décision a été décrite dans de nombreux ouvrages. Aussi, ce chapitre n'en fera qu'une courte présentation.

2.1 FONCTIONS ET FORMULES BOOLEENNES

On note B l'ensemble {0, 1} des *valeurs booléennes* (0 pour faux et 1 pour vrai). Les *variables booléennes* prennent leurs valeurs dans B et sont notées par des lettres latines en italique, *x*, *y*, *z*, éventuellement indicées : x_1, x_2,... Les ensembles de variables booléennes sont notés par des lettres latines majuscules en italique : *X*, *Y*, *Z*.

Définition 1 : Une *fonction booléenne f* sur un ensemble de variables booléennes $X = \{x_1,..., x_n\}$ est une fonction de B^n dans B .

Soit $X = \{x_1,..., x_n\}$ un ensemble de variables booléennes (supposé ordonné). Une *affectation* de *X* est un élément σ de B^n. On note $\sigma[x_i]$ la valeur de la variable booléenne x_i dans l'affectation σ (c'est-à-dire la ième composante du vecteur σ).

Les fonctions booléennes sont des objets mathématiques abstraits. Pour les manipuler concrètement, on utilise une syntaxe (ou, autrement dit, des structures de données) : les formules booléennes.

Définition 2 : L'ensemble F des *formules booléennes* est construit par induction structurelle à partir des deux constantes booléennes 0 et 1, d'un ensemble fini ou dénombrable de variables booléennes *X*, des connecteurs logiques usuels $\wedge$ (et), $\vee$ (ou), $\neg$ (négation) et des parenthèses (et). C'est le plus petit ensemble tel que :

- 0 et 1 sont des formules.
- Les variables de *X* sont des formules.
- Si *f* et *g* sont des formules, alors $f \wedge g$, $f \vee g$, $\neg$f et $\neg$g sont aussi des formules.

Les parenthèses sont utilisées pour délimiter les sous-formules.

Par exemple, $f = (x \wedge y) \vee (\neg x \wedge z)$ est une formule booléenne construite à partir de l'ensemble de variables $X = \{x, y, z\}$.

Le lien entre fonctions et formules booléennes est fait via les tables de vérité des connecteurs $\wedge$, $\vee$ et $\neg$. On étend la notion d'affectation aux formules. Soient σ une affection sur *X* et *f* et *g* deux formules construites sur

X. On pose σ[1]=1, σ[0]=0. La valeur d'une formule est calculée récursivement à partir de la valeur de ses sous-formules comme indiqué par la table suivante :

f	g	$\neg f$	$\neg g$	$f \wedge g$	$f \vee g$	$f \Rightarrow g$	$f \Leftrightarrow g$
1	1	0	0	1	1	1	1
1	0	0	1	0	1	0	0
0	1	1	0	0	1	1	0
0	0	1	1	0	0	1	1

Tableau 2-1 : Table de vérité de différentes formules usuelles

Une affectation σ *satisfait* la formule *f* (respectivement *falsifie*[1] *f*) si $f(\sigma) = 1$ (resp. $f(\sigma) = 0$). Une formule *f* est *satisfiable*[1] s'il existe une affectation qui la satisfait. Sinon elle est dite *insatisfiable*[1] (on dit aussi que c'est une *antilogie*). Si les 2^n affectations qu'il est possible de construire sur les *n* variables satisfont *f*, *f* est une tautologie.
La table ci-dessus comporte deux autres connecteurs logiques l'implication ⇒ et la bi-implication (ou équivalence) ⇔. Ces deux connecteurs peuvent être vus comme des raccourcis d'écriture. Ainsi $f \Rightarrow g$ est équivalent à $\neg f \vee g$ et $f \Leftrightarrow g$ est équivalent à $(f \Rightarrow g) \wedge (g \Rightarrow f)$.

D'autres connecteurs sont utiles dans le cadre des modèles booléens d'analyse du risque :
- L'opérateur « *k* sur *n* », noté $@k(f_1, ..., f_n)$, où $f_1, .., f_n$ sont des formules. Une affectation σ satisfait $@k(f_1, ..., f_n)$ si elle satisfait au moins *k* des f_i.
- L'opérateur de cardinalité, noté $\#(l,h)(f_1, ..., f_n)$, où $f_1, .., f_n$ sont des formules. Une affectation σ satisfait $\#(l,h)(f_1, ..., f_n)$ si elle satisfait au moins *l* et au plus *h* des f_i.
- L'opérateur « si-alors-sinon », noté *ite*(*f*,*g*,*h*) (pour *If-Then-Else)*. Il est équivalent à la formule $(f \wedge g) \vee (\neg f \wedge h)$. Il peut être lu : si *f* est vrai alors *g* sinon *h*.

Tous les connecteurs mentionnés jusquà présent travaillent sur un ensemble de variables *X*. Il est parfois utile de définir des opérateurs de projection : Soient $Y = \{x_1, ..., x_n, y\}$ un ensemble de variables booléennes et *f* une formule construite sur *Y*.
- Le cofacteur positif (respectivement négatif) de *f* par rapport à *y*, noté $f_{y=1}$ (respectivement $f_{y=0}$) est une formule telle que pour toute affectation σ de $X = \{x_1, ..., x_n\}$, σ satisfait $f_{y=1}$ si et seulement si $\sigma[y=1]$ (respectivement $\sigma[y=0]$) satisfait *f*, où $\sigma[y=v]$ dénote l'affectation de *Y* donnant la même valeur que σ aux x_i et la valeur *v* à *y*.
- Le quantificateur existentiel de *y* dans *f*, noté $\exists y\, f$ est une formule équivalente à la formule $f_{y=1} \vee f_{y=0}$.
- Le quantificateur universel de *y* dans *f*, noté $\forall y\, f$ est une formule équivalente à la formule $f_{y=1} \wedge f_{y=0}$.

2.2 ALGEBRES DE BOOLE

Soit E un ensemble contenant au moins deux éléments distinct 0 et 1, muni de deux lois internes binaires (deux applications de E×E dans E) notées traditionnellement « + » et « . » et d'une loi interne unaire notée traditionnellement « - ». Le sextuplet <E,0,1,+,.,-> forme une *algèbre de Boole* si les conditions suivantes sont réunies :
- + et . sont commutatives : $\forall a,b \in E$, a+b=b+a et a.b=b.a.
- + et . sont associatives : $\forall a,b,c \in E$, a+(b+c)=(a+b)+c et a.(b.c)=(a.b).c
- + et . sont distributives l'une par rapport à l'autre : $\forall a,b,c \in E$, a+(b.c)=(a+b).(a+c) et a.(b+c)=(a.b)+(a.c).
- 0 est un élément neutre pour + : ∀a 0+a=a+0=a et un élément absorbant pour . : ∀a a.0=0.a=0.
- 1 est un élément neutre pour . : ∀a 1.a=a.1=a et un élément absorbant pour + : ∀a a+1=1+a=1.
- Pour tout élément a, a+(-a)=1 et a.(-a)=0.

[1] Les verbes "satisfaire" et "falsifier" sont couramment employés en logique mathématique en lieu et place respectivement de "rendre vrai" et "rendre faux". Le sens à attribuer aux néologismes "satisfiable" (ou "satisfaisable") et "insatisfiable" en découle.

Toute algèbre de Boole <E,0,1,+,.,-> vérifie les propriétés suivantes :
- Idempotence de + et . : $\forall a \in E$, a+a=a et a.a=a.
- Lois de De Morgan : $\forall a,b \in E$, -(a+b)=(-a).(-b) et -(a.b)=(-a)+(-b).

Soit A=<E,0,1,+,.,-> une algèbre de Boole. Un sous-ensemble B de E est une *base* de A si les deux propriétés suivantes sont vérifiées :
- Les éléments de B sont indépendants deux à deux $\forall a,b \in B$, a.b=0.
- Tout élément de E peut s'exprimer comme une somme d'éléments de B.

Toute algèbre de Boole admet une base unique. Les éléments de la base sont appelés les *atomes* de l'algèbre de Boole.

Le théorème de Stone, qui est le théorème le plus important sur les algèbres de Boole, affirme que deux algèbres de Boole ayant le même nombre d'atomes sont isomorphes.

Soit E un ensemble fini ou dénombrable et soit *P*(E) l'ensemble des parties de E. On vérifie facilement que le sextuplet <P(E),∅,E,∪,∩,C>, où ∪, ∩ et C dénote respectivement l'union, l'intersection et la complémentation, forme une algèbre de Boole. Les atomes de cette algèbre sont les singletons {a}, où a est un élément de E.

Soit *X* un ensemble de variables booléennes et soit *F* l'ensemble des formules booléennes construites sur *X*. Le sextuplet <*F*,0,1,∨,∧,¬> forme aussi une algèbre de Boole appelée calcul propositionnel sur *X*. Pour étudier plus en détail cette algèbre, quelques définitions supplémentaires sont nécessaires.

Soit $X = \{x_1, ..., x_n\}$ un ensemble de variables booléennes.
- Un *littéral* est soit une variable *x*, soit sa négation ¬*x*. *x* est un *littéral positif*, ¬*x* est un *littéral négatif*. *x* et ¬*x* sont dit *opposés*.
- Un *produit* est un ensemble de littéraux qui ne contient pas un littéral et son opposé. Un produit est assimilé à la conjonction de ses littéraux.
- Un *minterme* sur *X* est un produit qui contient, positivement ou négativement, toutes les variables de *X*. On note *Mintermes(X)* l'ensemble des mintermes qu'il est possible de construire sur *X*.

Les mintermes construits sur *X* et les affectations de *X* sont en correspondance bi-univoque : l'affectation σ correspond au minterme π si pour toute variable *x* de *X*, σ[*x*]=1 si et seulement si $x \in \pi$. Par abus de langage, on dira qu'un minterme satisfait une formule si l'affectation correspondante en fait de même. Toute formule booléenne *f* construite sur *X* peut donc être vue comme un sous-ensemble de *Mintermes(X)* : *f* est équivalente à la disjonction des mintermes la satisfaisant. C'est pourquoi on se permettra de noter l'ensemble des mintermes satisfaisant la fonction *f* par *Mintermes(f)*.

Il en découle que le sextuplet <*P*(*Mintermes(X)*),∅,*Mintermes(X)*,∪,∩,C> forme une algèbre de Boole. On obtient les correspondances suivantes :

Formules booléennes construites sur X	P(Mintermes(X))
0	∅
1	Mintermes(X)
f	{π satisfaisant f}
∨	∪
∧	∩
¬	C

Tableau 2-2 : Correspondance des différentes notions

Les mintermes sont donc les atomes du calcul propositionnel sur *X*. A ce titre, ils jouent un rôle important et nombre de notions se comprennent plus simplement lorsqu'elles sont présentées via les mintermes.

2.3 COUPES MINIMALES

Dans les modèles booléens d'analyse du risque, les variables représentent la survenue de pannes de composants élémentaires et les formules décrivent les pannes des systèmes en fonction de ces dernières. Cela a au moins deux conséquences :

- D'une part, il est intéressant de connaître les jeux minima de pannes de composants élémentaires qui induisent une panne du système étudié. C'est cette idée de jeu minimum que capture la notion de coupe minimale.
- D'autre part, il existe une asymétrie fondamentale entre les littéraux positifs (qui représentent la survenue de pannes) et les littéraux négatifs (qui représentent leur non-survenue et sont donc, en un sens, moins porteurs d'information). Pour cette raison, la notion classique d'implicant premier, qui formalise l'idée de solution minimale d'une formule booléenne, n'est pas suffisante dans le cadre de l'analyse du risque.

Soient f et g deux formules booléennes construites sur un ensemble X de variables. Si tous les mintermes de f sont des mintermes de g, on dit que *f implique g* et on note $f \models g$.
Un *implicant* de f est un produit π, tel que $\pi \models f$.
Un implicant est *premier* s'il est minimal pour l'inclusion, c'est-à-dire s'il n'existe pas de sous-ensemble strict de π qui soit un implicant de f. On note PI[f] l'ensemble des implicants premiers d'une formule f (PI pour « *Prime Implicants* »).

Toute formule est équivalente à la disjonction de ses implicants premiers.

Soit, par exemple, la formule $f=(a \wedge b) \vee (\neg a \wedge c)$ construite sur $X=\{a,b,c\}$. f admet les implicants suivants : $a \wedge b \wedge c$, $a \wedge b \wedge \neg c$, $\neg a \wedge b \wedge c$, $\neg a \wedge \neg b \wedge c$, $a \wedge b$, $\neg a \wedge c$ et $b \wedge c$. Parmi eux, seuls les trois derniers sont des implicants premiers.

Cet exemple permet d'illustrer la critique « fiabiliste » de la notion d'implicant premier. Le produit $\neg a \wedge c$ contient un littéral négatif ($\neg a$) qui n'est pas porteur d'une information bien intéressante. En fait, on ne voudrait garder de ce produit que c. Mais on a $b \wedge c \models c$ et donc $b \wedge c$ n'est pas minimal (ou premier). L'ensemble des coupes minimales que l'on souhaite obtenir est donc $\{a \wedge b, c\}$. Intuitivement, une coupe minimale est donc une partie positive minimale d'une affectation satisfaisant f (d'une solution de f) : si les pannes représentées par cette partie positive sont survenues et si aucun autre composant n'est en panne, alors le système est en panne. Il est surprenant de constater que, bien que les modèles booléens d'analyse du risque sont utilisés depuis de nombreuses années, la notion de coupes minimales n'a été formalisée que très récemment. En particulier, aucune des monographies de référence citées au début de ce chapitre n'en donne une définition précise. On peut pourtant donner une formalisation directe de l'intuition ci-dessus.

Soient X un ensemble de variables et π un produit (positif) sur X. On note π^c le minterme de X formé de π et des littéraux négatifs construits sur les variables n'apparaissant dans π.
Par exemple, si $X=\{a,b,c\}$ et $\pi=c$ alors $\pi^c=\neg a \wedge \neg b \wedge c$.

Soient X un ensemble de variables, f une formule et π un produit (positif) construits sur X.

- π est une *coupe* de f si π^c satisfait f.
- π est une *coupe minimale* s'il n'existe pas de sous-ensemble strict de π qui soit une coupe de f. On note MCS[f] l'ensemble des coupes minimales de f (MCS pour « *Minimal Cutsets* »)

La fonction $f = (a \wedge b) \vee (\neg a \wedge c)$ construite sur $X = \{a,b,c\}$ admet les coupes suivantes : $a \wedge b \wedge c$, $a \wedge b$, $a \wedge c$, $b \wedge c$ et c. Parmi elles, seules $a \wedge b$ et c sont minimales.

Pour expliciter le lien entre coupes minimales et implicants premiers, on introduit un ordre naturel sur les littéraux : pour toute variable x, on pose $x < \neg x$. Cet ordre sur les littéraux est ensuite étendu en un ordre sur les mintermes : soient π et ρ deux mintermes construits sur un ensemble de variables $X=\{x_1, ..., x_n\}$. $\pi < \rho$ si pour tout x_i, $\pi[x_i] \leq \rho[x_i]$ et s'il existe au moins un x_i pour lequel cette inégalité est stricte.

Une formule *f* est *monotone* (on dit aussi *cohérente*) si pour tous produits π et ρ on a : si $\pi \models f$ et $\rho < \pi$ alors $\rho \models f$. Une condition suffisante, mais pas nécessaire, pour qu'une formule soit monotone est qu'elle ne comporte que des variables et des connecteurs $\wedge$ et $\vee$.

Les implicants premiers d'une formule monotone ne contiennent que des littéraux positifs. En effet, si $\neg x \wedge \pi$ était un implicant premier d'une formule monotone *f*, alors $x \wedge \pi$ satisferait *f* (par définition de la monotonie) et donc π aussi.
Les notions d'implicant premier et de coupe minimale coïncident donc pour les formules monotones : pour toute formule monotone *f*, PI[*f*]=MCS[*f*]. Dans le cas général, les coupes minimales d'une formule *f* sont les implicants premiers d'une certaine formule monotone *g*. Si *f* est elle-même monotone, on a bien sûr *g*=*f*. Plus précisément, on définit l'*enveloppe monotone* $\uparrow f$ d'une fonction *f* construite sur un ensemble de variables *X* de la façon suivante :

$$\uparrow f = \{\pi \in \mathit{Mintermes}(X); \exists \rho \in \mathit{Mintermes}(X), \rho \text{ satisfait } f \text{ et } \pi \leq \rho\} \qquad [2\text{-}1]$$

Il est clair que pour toute fonction *f*, $f \models \uparrow f$. On a de plus le résultat suivant :

$$\mathrm{MCS}[f] = \mathrm{PI}[\uparrow f]. \qquad [2\text{-}2]$$

Pour finir cette présentation de la notion de coupe minimale, il paraît important de rappeler quelques résultats sur la complexité théorique de la recherche et du comptage des implicants premiers.

- On peut construire 3^n produits sur un ensemble de *n* variables (chaque variable étant soit absente, soit présente positivement, soit présente négativement). Le nombre maximum d'implicants premiers d'une formule est en $O(3^n / \sqrt{n})$ [Qui52,CM78].
- On peut construire 2^n produits positifs sur un ensemble de *n* variables. Le nombre maximum de coupes minimales d'une formule croît aussi exponentiellement avec le nombre de variables.
- Compter le nombre d'implicants premiers d'une formule, même monotone, est un problème #P-difficile. La classe de complexité #P a été introduite par Valiant pour caractériser les problèmes de comptage associés aux problèmes de décision NP-difficiles [Val79a,Val79b].
- On peut construire $2^k \times C_n^k$ produits contenant exactement *k* littéraux sur un ensemble de *n* variables. Ce nombre est polynomial en *n*. Déterminer si une formule admet un implicant premier contenant moins de *k* littéraux est un problème NP-difficile [Pap94]. En revanche, déterminer toutes les coupes minimales d'une formule contenant moins de *k* variables est un problème de complexité polynomiale [Rau01a]. Ce dernier point est argument très fort en faveur de la notion de coupe minimale.

En pratique, les formules issues d'étude de sûreté de fonctionnement admettent souvent un nombre gigantesque de coupes minimales et un nombre plus grand encore d'implicants premiers. On est en revanche capable de déterminer relativement efficacement leurs coupes minimales « courtes » qui sont les plus probables.

2.4 FORMULES BOOLEENNES ET PROBABILITES

Un des objectifs des études d'analyse du risque est d'évaluer les probabilités d'occurrence des événements redoutés et de mesurer la contribution des différents composants à ces probabilités. La probabilité d'occurrence d'un événement peut être obtenue de deux manières : par une approche expérimentale ou par calcul à partir des probabilités d'événements plus élémentaires.

L'approche expérimentale consiste à approcher la probabilité en calculant une fréquence relative à partir d'essais : si on a réalisé un nombre *N* suffisamment grand d'expériences dans des conditions identiques et que l'on a observé *n* fois l'événement *e*, le rapport *n/N* donne une bonne approximation de la probabilité recherchée *P*(*e*). En effet, cette probabilité peut être formellement définie comme suit :

$$P(e) = \lim_{N \to \infty} (n / N) \qquad [2\text{-}3]$$

Lorsque l'événement considéré est décrit par une fonction booléenne d'événements élémentaires, sa probabilité doit être calculée en combinant les probabilités de ces derniers. Avant de décrire les deux principales familles de méthodes utilisées en pratique pour ce faire, quelques définitions et rappels sont nécessaires.

Deux événements sont *mutuellement exclusifs* ou incompatibles s'ils ne peuvent pas apparaître simultanément. Si des événements sont mutuellement exclusifs, la probabilité d'occurrence d'au moins l'un d'entre eux est la somme de leurs probabilités :

$$P(x \vee y) = P(x) + P(y) \qquad [2\text{-}4]$$

Si les événements ne sont pas exclusifs, le résultat précédent doit être modifié pour prendre en compte le cas où les deux événements se produisent :

$$P(x \vee y) = P(x) + P(y) - P(x \wedge y) \qquad [2\text{-}5]$$

Cette propriété se généralise à *n* événements (*Théorème de Sylvester-Poincaré*) :

$$\begin{aligned} P(x_1 \vee \ldots \vee x_n) = & \sum_{i=1}^{n} P(x_i) - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P(x_i \wedge x_j) \\ & + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} P(x_i \wedge x_j \wedge x_k) - \ldots \\ & + (-1)^{n+1} P(x_1 \wedge \ldots \wedge x_n) \end{aligned} \qquad [2\text{-}6]$$

Deux événements sont *mutuellement indépendants* lorsque l'occurrence de l'un n'affecte pas la probabilité d'occurrence de l'autre. Si des événements sont mutuellement indépendants, la probabilité de leur occurrence simultanée est le produit de leur probabilité :

$$P(x \wedge y) = P(x) \times P(y) \qquad [2\text{-}7]$$

Si les événements sont dépendants, l'occurrence du second événement est conditionnée par l'occurrence du premier événement. Ceci se traduit à l'aide du théorème des *probabilités conditionnelles* :

$$P(x \wedge y) = P(x) \times P(y / x) = P(y) \times P(x / y) \qquad [2\text{-}8]$$

Où *P*(*x*/*y*) dénote la probabilité que l'événement *x* se produise sachant que l'événement *y* s'est déjà produit.

Les méthodes booléennes d'analyse du risque (qui seront décrites au chapitre II) font l'hypothèse que les variables apparaissant dans les formules codent des événements indépendants (le traitement des fameuses défaillances de cause commune relevant plus d'une approche empirique que d'une analyse mathématique rigoureuse). Si cette hypothèse est vérifiée, il est, au moins théoriquement, possible de calculer la probabilité d'une formule *f* à partir des probabilités de ses mintermes. Par définition, ces derniers sont exclusifs deux à deux. On a donc :

$$P(f) = \sum_{p \in Mintermes(f)} P(p) = \sum_{p \in Mintermes(f)} \prod_{x \in p} P(x) \qquad [2\text{-}9]$$

Cette méthode est très inefficace car elle demande d'expliciter tous les mintermes de *f*.

La plupart des logiciels de traitement calculent la probabilité de la formule en appliquant le développement de Sylvester-Poincaré sur les coupes minimales de cette dernière. Plus précisément, ils calculent une approximation de cette probabilité en faisant trois types de simplifications :

1. Ils considèrent l'enveloppe monotone de la formule plutôt que la formule elle-même. Cette simplification est pessimiste dans la mesure où pour toute formule f, $f \models \uparrow f$ et donc $P(f) \leq P(\uparrow f)$.
2. Ils ne considèrent que les coupes minimales « prépondérantes » de f (celles de plus forte probabilité). Cette simplification est au contraire optimiste puisqu'elle revient à ignorer une partie des mintermes de f.
3. Ils ne considèrent que les k premiers termes du développement de Sylvester-Poincaré, voire le premier seulement. Cette simplification peut être pessimiste (si k est impair) ou optimiste (si k est pair). Elle repose sur l'asymétrie entre les littéraux positifs et négatifs : dans la mesure où les systèmes étudiés sont en général très sûrs (issus typiquement de l'industrie nucléaire), les probabilités des littéraux positifs sont très inférieures à celle des littéraux négatifs. Une coupe minimale de grande taille est donc très improbable et peut être ignorée.

Le chapitre IV présente un étude détaillée sur la validité de ces simplifications dans une application industrielle de grande taille.

L'autre technique utilisée en pratique pour calculer la probabilité d'une formule à partir des probabilités de ses variables donne des résultats exacts. Elle repose sur la décomposition de Shannon.
Soit f une fonction booléenne dépendant d'un ensemble de variables X et x une variable appartenant à X. La *décomposition de Shannon* de f par rapport à x est donné par :

$$f = (\neg x \wedge f_{x=0}) \vee (x \wedge f_{x=1}) \qquad [2\text{-}10]$$

Le premier et le second termes de la disjonction sont mutuellement exclusifs. On a donc :

$$P(f) = \left[(1 - P(x)) \times P\left(f_{x=0}\right)\right] + \left[P(x) \times P\left(f_{x=1}\right)\right] \qquad [2\text{-}11]$$

Les diagrammes binaires de décision (qui seront présentés plus loin), permettent de manipuler efficacement les formules sous forme de Shannon.

D'un point de vue théorique (et pratique) déterminer la probabilité exacte d'une formule est un problème difficile (#P-difficile [Val79b]). De plus, il n'y a pas d'espoir de trouver des algorithmes efficaces donnant de bonnes approximations de cette probabilité. En effet, Rosenthal a montré que pour tout réel $\rho > 1$, trouver une valeur qui soit dans l'intervalle $[P(f)/\rho, P(f).\rho]$ est un problème NP-difficile [Ros75].

2.5 L'ALGORITHME MOCUS

La plupart des logiciels de traitement d'arbres de défaillance et d'arbres d'événements disponibles aujourd'hui utilisent une variante de l'algorithme MOCUS pour déterminer les coupes minimales des formules considérées. Cet algorithme a été proposé originellement par Fussel et Vesely [FV72]. Son principe est en fait très proche de celui d'autres méthodes utilisées en démonstration automatique, comme la méthode des tableaux (voir, par exemple, [GG90] pour une présentation de cette méthode). L'article [FV72] ne propose en fait qu'un schéma d'algorithme. En particulier, les structures de données utilisées ne sont pas décrites et de nombreux détails restent dans l'ombre. C'est pourquoi MOCUS doit être vu comme le nom générique d'une famille d'algorithmes, plutôt que comme un algorithme particulier. La présentation qui est faite dans cette section suit l'article [Rau02a] qui décrit la variante de MOCUS mise en œuvre dans Aralia.

MOCUS suppose que la formule f dont on veut déterminer les coupes minimales est donnée sous forme d'un arbre ET/OU. Un arbre ET/OU est une formule construite à partir de littéraux positifs et négatifs, de conjonctions et de disjonctions. En d'autres termes, les négations ont été repoussées le plus bas possible dans la formule. Il existe un algorithme très simple pour transformer toute formule en un arbre ET/OU équivalent [Rau02a]. Cet algorithme est de plus de complexité linéaire en la taille de la formule de départ (si celle-ci ne contient pas des portes k sur n portant sur un nombre non borné de sous-formules).
Il est pratique de représenter les arbres ET/OU sur un ensemble de variables X par des ensembles d'équations booléennes de la forme $g = \mathrm{OP}(p_1,...,p_r)$, où g est une variable booléenne n'appartenant pas à X, OP est un opérateur réalisant une conjonction ($\wedge$) ou une disjonciton ($\vee$) sur les p_i qui sont soit des variables

apparaissant en membre gauche d'une équation, soit des littéraux construits sur X. Chaque variable g est associée un des connecteurs de la formule. Elle apparaît donc une et une seul fois en tant que membre gauche d'une équation. Une des variables g est associée avec le connecteur sommet de la formule. Par exemple, l'arbre ET/OU $(a \wedge b) \vee (\neg a \wedge c)$ est représenté par l'ensemble d'équations suivant :

$g_{top} = g_1 \vee g_2$
$g_1 = a \wedge b$
$g_2 = \neg a \wedge c$

L'algorithme maintient deux ensembles de produits : l'ensemble T des produits restant à traiter et l'ensemble C des coupes minimales déjà trouvées. Son principe général est le suivant :

1 T ← { g_{top} }, C ← ∅
2 tant que T ≠ ∅ faire
3 choisir un produit π de T, retirer π de T
4 si π ne contient que des littéraux construits sur X
5 alors extraire les coupes minimales de π et les ajouter à C
6 sinon choisir une variable g de π n'appartenant pas à X (π = $g \wedge$ π')
7 soit g = OP(p_1, ... ,p_r) l'équation dont le membre gauche est g
8 si OP est un ET
9 alors ajouter le produit $p_1 \wedge ... \wedge p_r \wedge$ π' à T
10 sinon ajouter les produits $p_1 \wedge$ π',..., $p_r \wedge$ π' à T
11 fait

Pour rendre effectif l'algorithme décrit ci-dessus, plusieurs points doivent impérativement être précisés :

1. L'ensemble T doit être géré de façon à pouvoir ajouter et retirer efficacement un produit. En particulier, à chaque ajout d'un produit π on doit vérifier qu'il n'existe pas déjà un produit ρ dans T tel que $\rho \subseteq \pi$ (auquel cas π ne doit pas être ajouté à T). De plus, on doit supprimer de T tous les produits σ tels que $\pi \subset \sigma$.
2. On doit définir des heuristiques de choix du prochain produit π à traiter (ligne 3) et de la prochaine variable g à substituer (ligne 6).
3. Finalement, on doit définir un algorithme pour extraire les coupes minimales d'un produit terminal π (ligne 5). Dans Aralia, cela est fait de la façon suivante : on considère les mintermes ρ^c, où ρ est obtenu en supprimant un littéral positif de π. Si aucun de ces mintermes ne satisfait f, c'est que la partie positive de π est une coupe minimale de f. Sinon, on applique le principe récursivement sur les ρ tels que ρ^c satisfait f.

Le succès de l'algorithme MOCUS est en grande partie dû à la possibilité de simplifier les calculs en ne considérant que les coupes minimales prépondérantes, c'est-à-dire les plus probables. En effet, si la probabilité du produit π en cours de traitement (plus précisément, le produit des probabilités des littéraux positifs construits sur X de π) est inférieure à un certain seuil S, alors le traitement de π peut être abandonné (une preuve formelle de cette propriété est établie dans [Rau02a]). Le seuil S peut être défini en valeur absolue ou relativement à la somme des probabilités des coupes minimales déjà obtenues.

L'article [Rau02a] propose aussi la notion de « *shadow variables* » qui permet de traiter efficacement les branches de succès des arbres d'événements (voir chapitre 2).

2.6 LES DIAGRAMMES BINAIRES DE DECISION

Les diagrammes binaires de décision de Bryant [Bry86,BRB90] constituent la structure de données la plus efficace connue à ce jour pour coder et manipuler des fonctions booléennes. Dans la suite de ce document, l'acronyme anglais BDD sera utilisé (pour « *Binary Decision Diagrams* ») pour désigner cette technique à laquelle de nombreux articles et ouvrages ont été consacrés (voir, par exemple, [Bry92,Min96,MT98] pour des présentations générales). Cette section se contentera d'en donner un bref aperçu. Depuis leur introduction dans le domaine de la sûreté de fonctionnement [CM92,Rau93], les BDD se sont effectivement avérés être la

technique la plus efficace pour traiter les modèles booléens d'analyse du risque. Le logiciel Aralia a joué et joue toujours un rôle pionnier dans ce domaine.

2.6.1 DEFINITIONS ET PRINCIPES

Le BDD associé à une formule booléenne est un codage compact de la table de vérité de cette dernière. Ce codage est fondé sur la décomposition de Shannon des formules booléennes.
Soit *f* une formule booléenne dépendant d'une variable *x*. L'égalité suivante est vérifiée :

$$f = (\neg x \wedge f_{x=0}) \vee (x \wedge f_{x=1}) \qquad [2\text{-}12]$$

En choisissant un ordre total sur les variables et en appliquant récursivement la décomposition de Shannon, la table de vérité de toute formule peut être représentée graphiquement par un arbre binaire. Chaque nœud interne de cet arbre code une formule *f* et peut se lire comme un opérateur *si-alors-sinon*. Il est étiqueté par une variable *x* et il a deux arêtes sortantes (une arête pointant sur le nœud codant le cofacteur positif $f_{x=1}$ de *f* par rapport à *x* et une autre pointant sur le nœud codant le cofacteur négatif). Les feuilles de l'arbre sont étiquetées par 0 ou par 1. La valeur de la formule pour une certaine affectation est obtenue en descendant le long de la branche correspondante à partir du nœud racine de l'arbre jusqu'à une feuille. L'arbre de Shannon pour la formule $(a \wedge b) \vee (\neg a \wedge c)$ et pour l'ordre alphabétique est dessiné ci-dessous (les arêtes en pointillés sont les arêtes *sinon*)

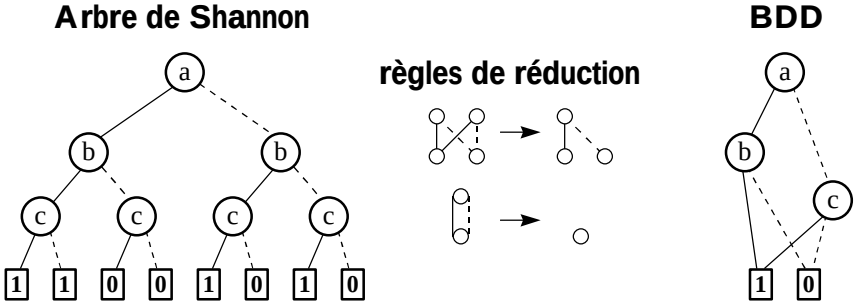


Figure 2.1 : Transformation d'un arbre de Shannon en BDD

Bien sûr, une telle représentation est très coûteuse en place mémoire. Il est toutefois possible de la réduire considérablement en utilisant les deux règles suivantes :

1. Le partage des sous-arbres isomorphes. Puisque deux sous-arbres isomorphes codent la même fonction, un seul suffit. En particulier, on aura besoin que de deux feuilles (une étiquetée par 0, l'autre par 1).
2. La suppression des nœuds inutiles. Un nœud dont les deux arêtes sortantes pointent sur le même nœud code la même fonction que ce dernier ($f = (\neg x \wedge g) \vee (x \wedge g)$). Un tel nœud est donc inutile.

En appliquant ces deux règles aussi souvent que nécessaire, on obtient le BDD associé à la formule. Le processus de réduction est illustré sur la partie droite de la figure ci-dessus.
Un BDD est donc un graphe orienté acyclique. Pour un ordre des variables donné, le BDD codant une formule est unique, à un isomorphisme près [Bry86]. On dit que les BDD sont une représentation canonique des formules booléennes. Cette unicité à de nombreuses conséquences intéressantes. Par exemple, tester si deux formules sont équivalentes, lorsqu'on dispose des BDD codant ces formules, revient à tester que les nœuds racines des deux BDD sont codés à la même adresse mémoire.

Les opérations logiques (conjonction, disjonction, négation,…) peuvent être effectuées directement sur les BDD. Cela résulte de l'orthogonalité des connecteurs usuels avec la décomposition de Shannon. Pour toutes fonctions *f* et *g* et tout connecteur binaire ⊗, l'égalité suivante est vérifiée :

$$\begin{gathered}[(\neg x \wedge f_{x=0}) \vee (x \wedge f_{x=1})] \otimes [(\neg x \wedge g_{x=0}) \vee (x \wedge g_{x=1})] \\ = \\ (\neg x \wedge [f_{x=0} \otimes g_{x=0}]) \vee (x \wedge [f_{x=1} \otimes g_{x=1}])\end{gathered} \qquad [2\text{-}13]$$

En d'autres termes, pour calculer le BDD codant la formule $f \otimes g$ à partir des BDD codant *f* et *g*, on applique l'opérateur ⊗ sur les fils gauches, puis sur les fils droits des nœuds racines et on compose les deux résultats.

Les cas terminaux de cet algorithme récursif sont donnés par les règles classiques de simplification des formules booléennes, par exemple $f \wedge 1 = f$, $f \wedge 0 = 0$, $f \wedge f = f$, …

Pour obtenir le BDD codant une formule, on ne passe donc jamais par l'arbre de Shannon. Ce BDD est obtenu en composant les BDD des sous-formules. De plus, un mécanisme de cache est utilisé : chaque fois qu'une opération est réalisée, on met son résultat dans une table. Avant d'effectuer une opération, on va donc d'abord voir dans la table si son résultat n'est pas déjà connu. Ce mécanisme astucieux rend les opérations classiques sur les BDD (disjonction, conjonction,…) polynomiales en la taille de leur opérandes. La négation peut même être effectuée en temps constant en ajoutant des drapeaux sur les arêtes. La description complète d'un paquetage BDD est donnée dans la référence [BRB90].
La table suivante donne la complexité, dans les plus mauvais cas, des principales opérations logiques sur les BDD ($|f|$ dénote la taille du BDD codant f)

Opération	Complexité
f = g ?	O(1)
$f \wedge g$, $f \vee g$, $f \Rightarrow g$, $f \Leftrightarrow g$,…	$O(\|f\| \times \|g\|)$
$f_{x=v}$	$O(\|f\|)$
$\exists x\ f$, $\forall x\ f$	$O(\|f\|^2)$

Tableau 2-3 : Complexité des principales opérations logiques sur les BDD

La plupart des opérations sur les BDD sont complètement décrites par des équations récursives comme celles décrivant le principe de calcul des opérations logiques (en supposant implicitement que le mécanisme de cache est mis en œuvre). Par exemple, l'algorithme calculant la probabilité d'une formule à partir de la probabilité des variables et du BDD codant la formule est décrit par les trois équations suivantes :

$$P(1) = 1 \qquad [2\text{-}14]$$
$$P(0) = 0$$
$$P((\neg x \wedge f_{x=0}) \vee (x \wedge f_{x=1})) = [(1-P(x)) \times P(f_{x=0})] + [P(x) \times P(f_{x=1})]$$

Grâce au mécanisme de cache, cet algorithme est linéaire en la taille du BDD.

2.6.2 TAILLE DES BDD ET ORDRE DES VARIABLES

Un résultat classique de la théorie de l'information affirme que presqu'aucune fonction booléenne construite sur n variables n'admet un circuit (c'est-à-dire une formule la représentant) de taille polynomiale en n. Ceci est *a fortiori* vrai pour les BDD. Cela étant, ce résultat n'a que peu d'intérêt pratique puisque les fonctions sont toujours manipulées via des formules dont la taille doit être raisonnable (et donc polynomiale par rapport au nombre de variables).
En fait, depuis le début de l'utilisation des BDD, on sait que :

- La taille des BDD dépend fortement de l'ordre choisi sur les variables.
- Il existe des fonctions naturelles qui admettent un circuit de taille polynomiale, mais pas de BDD de taille polynomiale, quel que soit l'ordre choisi sur les variables [Bry91].
- Il n'existe pas de caractérisation syntaxique simple des formules admettant des BDD de taille polynomiale.

L'exemple donné dans [Bry86] illustre le problème de l'ordre des variables. Soit f_n la formule paramétrique suivante :

$$f_n = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee ... \vee (a_n \wedge b_n) \qquad [2\text{-}15]$$

La taille du BDD codant f_n est exponentielle en n pour l'ordre $a_1<a_2<...<a_n<b_1<...<b_n$ et linéaire pour l'ordre $a_1<b_1<a_2<b_2<...<a_n<b_n$.

Trouver un bon ordre sur les variables est un problème difficile. En effet, dans la plupart des cas, on ne connaît pas d'autre moyen pour prédire la taille d'un BDD que de le construire. On utilise donc des heuristiques pour trouver des ordres acceptables. Ces heuristiques dépendent du domaine étudié. Plusieurs

études ont ainsi porté sur les modèles booléens d'analyse du risque (voir, par exemple [BBR97,AB98]). Ces heuristiques reposent sur des principes différents. Elles s'efforcent néanmoins toutes de rendre proche dans l'ordre les variables qui sont proches dans la formule (comme les a_i et b_i de l'exemple ci-dessus).
Le livre de Wegener [Weg00] donne une vue complète des résultats connus sur la taille des BDD, l'influence de l'ordre des variables.

2.6.3 ZERO-SUPPRESSED BDDS

Pour travailler avec les coupes minimales et les implicants premiers, on a besoin de coder des ensembles de produits. Les BDD ne peuvent pas servir directement à coder de tels ensembles. En effet, une variable peut apparaître positivement, négativement ou pas du tout dans un produit. Il faudrait donc des diagrammes ternaires de décision (ce qui a d'ailleurs été proposé dans [Sas96]). Une autre solution, qui s'est imposée, consiste à utiliser les BDD, mais en leur donnant une autre sémantique. C'est la notion de « *Zero-Suppressed BDD* » (ZBDD) proposée par Minato [Min93]. L'idée de Minato est d'étiqueter les nœuds par des littéraux et de décomposer les ensembles de produits en fonction de la présence ou de l'absence d'un littéral :

- Les feuilles *0* et *1* codent respectivement l'ensemble vide de produits et l'ensemble ne contenant que le produit vide.
- Un nœud interne N étiqueté par le littéral q et pointant vers les nœuds N_1 (fils alors) et N_0 (fils sinon) code l'ensemble de produits $S=\{\{q\} \cup \pi\ ;\ \pi \in S_1\} \cup S_0$, où S_1 et S_0 sont respectivement les ensembles de produits codés par N_1 et N_0.

Cette représentation des ensembles de produits est canonique. Elle nécessite de changer la deuxième règle de réduction des BDDs qui devient : les nœuds dont le fils gauche (branche alors) est la feuille *0* sont inutiles.

Les opérations ensemblistes (union, intersection, différence) se programment sur les ZBDD de façon similaire aux opérations logiques sur les BDD. Elles utilisent le même mécanisme de cache. Leur complexité dans le pire des cas est donc le produit des tailles de leurs opérandes.

Grâce au partage des sous-arbres isomorphes, les BDD et les ZBDD rendent possibles le codage et la manipulation de très grosses fonctions (ou de très gros ensembles de produits). Bien sûr, ces techniques ne constituent pas la panacée universelle, mais elles ont permis un saut technologique important.

2.6.4 THEOREMES DE DECOMPOSITION

Pour passer du BDD codant une fonction au ZBDD codant ses coupes minimales (ou ses implicants premiers), on a besoin d'un algorithme. C'est-à-dire, dans le cadre des BDD, d'équations récursives définissant cet algorithme. Ces équations sont le résultat de théorèmes de décomposition.

Soient q un littéral et S *et* T deux ensembles de produits. On suppose de plus que ni q, ni son opposé n'apparaissent dans S. On note :

- S/T la différence ensembliste de S et T.
- $q.S$ l'ensemble de produits suivant : $q.S = \{\{q\} \cup \pi\ ;\ \pi \in S\}$.
- $S \div T$ l'ensemble de produits suivant : $S \div T = \{\pi \in S, \forall \rho \in T, \rho \neq \pi \text{ et } \rho \not\subset \pi\}$

Le premier théorème de décomposition concerne les implicants premiers. Il est dû à Morreale [Mor70].

Soit f une fonction booléenne et soit x une variable dont f dépend. Alors l'ensemble PI[f] des implicants premiers de f est l'union des trois ensembles P_2, P_1, P_0 suivants :

- $P_2 = \text{PI}[f_{x=0} \wedge f_{x=1}]$
- $P_1 = x.(\text{PI}[f_{x=1}] / P_2)$
- $P_0 = \neg x.(\text{PI}[f_{x=0}] / P_2)$

Les deux théorèmes suivants donnent deux façons différentes de calculer les coupes minimales. Ils ont été proposés dans [DR97].

Soit f une fonction booléenne et soit x une variable dont f dépend. Alors l'ensemble MCS[f] des coupes minimales de f est l'union des deux ensembles P_1, P_0 suivants :
Premier théorème :

- $P_0 = \text{MCS}[f_{x=0}]$
- $P_1 = x.(\text{MCS}[f_{x=0} \vee f_{x=1}] / P_0)$

Second théorème :

- $P_0 = \text{MCS}[f_{x=0}]$
- $P_1 = x.(\text{MCS}[f_{x=1}] \div P_0)$

Ces trois théorèmes peuvent en fait être réunis en un seul [Rau01a]. Ils permettent de définir un algorithme général de calcul des coupes minimales et des implicants premiers d'une fonction (voire de mélanger les deux notions).

2.7 BIBLIOGRAPHIE

[AB98] J.D. Andrews and L.M. Barlett. *Efficient Basic Event Orderings for Binary Decision Diagrams*. In Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'98, pages 61–67, 1998. ISSN 0149-144X.

[AM93] J.D. Andrews and T.R. Moss. *Reliability and Risk Assessment*. John Wiley & Sons, 1993. ISBN 0-582-09615-4.

[BBR97] M. Bouissou, F. Bruyère, and A. Rauzy. *BDD based Fault-Tree Processing: A Comparison of Variable Ordering Heuristics*. In C. Guedes Soares, editor, Proceedings of European Safety and Reliability Association Conference, ESREL'97, volume 3, pages 2045–2052. Pergamon, 1997. ISBN 0-08-042835-5.

[BC01] T. Bedford and R. Cook. *Probabilistic Risk Analysis. Foundations and Methods*. Cambridge University Press. ISBN 0521-77320-2. 2001.

[BRB90] K. Brace, R. Rudell, and R. Bryant. *Efficient Implementation of a BDD Package*. In Proceedings of the 27th ACM/IEEE Design Automation Conference, pages 40–45. IEEE 0738, 1990.

[Bry86] R. Bryant. *Graph Based Algorithms for Boolean Function Manipulation*. IEEE Transactions on Computers, 35(8):677–691, August 1986.

[Bry91] R. Bryant. *On the Complexity of VLSI Implementations and Graphs Representations of Boolean Functions with Application to Integer Multiplication*. IEEE Transactions on Computers, 40(2):205–213, 1991.

[Bry92] R. Bryant. *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*. ACM Computing Surveys, 24:293–318, September 1992.

[CM78] A.K. Chandra and G. Markowsky. *On the number of prime implicants*. Discrete Mathemathics, 24:7–11, 1978.

[CM92] O. Coudert and J.-C. Madre. *A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions*. In T. Knight and J. Savage, editors, Advanced Research in VLSI and Parallel Systems, pages 113–128, March 1992.

[Coc97] C. Cocozza-Thivent. *Processus stochastiques et fiabilité des systèmes*. Springer Verlag, 1997. ISBN 3-540-63390-1.

[DR97] Y. Dutuit and A. Rauzy. *Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia*. Reliability Engineering and System Safety, 58:127–144, 1997.

[FV72] J.B. Fussel and W.E. Vesely. *A New Methodology for Obtaining Cut Sets for Fault Trees*. Trans. Am. Nucl. Soc., 15:262–263, June 1972.

[GG90] P. Gochet and P. Gribomont. *Logique, méthodes pour l'informatique fondamentale, volume 1*. éditions Hermès, 1990. ISBN 2-86601-249-6.

[Lim91] N. Limnios. *Arbres de défaillance*. Traité des Nouvelles Technologies. Hermes, 1991.

[Min93] S. Minato. *Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems*. In Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93, pages 272–277, 1993.

[Min96] S.-I. Minato. *Binary Decision Diagrams and Applications to VLSI CAD*. Kluwer Academics Publishers, 1996. ISBN 0-7923-9652-9.

[Mor70] E. Morreale. *Recursive Operators for Prime Implicant and Irredundant Normal Form Determination*. IEEE Trans. Computers, C-19(6):504–509, 1970.

[MT98] C. Meinel and T. Theobald. *Algorithm and Data Structures in VLSI Design*. Springer Verlag, 1998. ISBN 3-540-64486-5.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. ISBN 0-201-53082-1.

[PG80] A. Pagès and M. Gondran. *Fiabilité des systèmes*, Collection de la Direction des études et Recherches d'électricité de France. Eyrolles, 1980. ISSN 0399-4198.

[Qui52] W.V.O. Quine. *The problem of simplifying truth functions*. American Mathematics Monthly, 59:521–531, 1952.

[Rau01a] A. Rauzy. *Mathematical Foundations of Minimal Cutsets*. IEEE Transactions on Reliability, 2001. To appear.

[Rau02a] A. Rauzy. *Towards an Efficient Implementation of Mocus*. IEEE Transactions on Reliability, 2001. To appear.

[Rau93a] A. Rauzy. *New Algorithms for Fault Trees Analysis*. Reliability Engineering & System Safety, 05(59):203–211, 1993.

[Ros75] A. Rosenthal. *A computer scientist looks at reliability computations*. In R.E. Barlow, J.B. Fussel, and N.D. Signpurwalla, editors, Reliability and fault tree analysis, pages 133–152. SIAM, 1975.

[Sas96] T. Sasao. *Ternary Decision Diagrams and their Applications*. In Representations of Discrete Functions, chapter 12, pages 269–292. Kluwer Academic Publisher, 1996. ISBN 0-7923-9720-7.

[Val79a] L.G. Valiant. *On the complexity of computing the permanent*. Theoretical Computer Science, 8:189–201, 1979.

[Val79b] L.G. Valiant. *The complexity of enumeration and reliability problems*. SIAM Journal of Computing, 8:410–421, 1979.

[Vil88] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, 1988.

[Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000. ISBN 0-89871-458-3.

3. MODELES BOOLEENS D'ANALYSE DU RISQUE

Dans ce chapitre, nous présentons les deux méthodes booléennes les plus fréquemment utilisées dans le domaine de l'analyse des risques et, d'une manière plus générale, en sûreté de fonctionnement. Il s'agit des réseaux de fiabilité, englobant les diagrammes de fiabilité, et des arbres de défaillance.
Une des caractéristiques majeures communes de ces méthodes est qu'elles considèrent l'état du système à tout instant t comme une combinaison des états, au même instant, de ses éléments constitutifs, indépendamment de l'ordre dans lequel ces éléments y sont arrivés.
Elles se différencient cependant par le fait que les réseaux et diagrammes de fiabilité modélisent, d'une certaine façon, la logique de fonctionnement des systèmes étudiés, alors que les arbres de défaillance modélisent leur logique de dysfonctionnement. De plus, la démarche de construction des premiers est de nature inductive (des causes vers les conséquences), tandis que les seconds s'élaborent d'une manière déductive (des conséquences ou pannes vers les causes).
Une autre caractéristique commune à ces modèles est qu'ils peuvent servir de support à une double évaluation des performances du système étudié :

- qualitative, par la détermination des scénarios de bon fonctionnement ou de panne,
- quantitative, par la mesure probabiliste de ces mêmes performances obtenues soit par un traitement de nature combinatoire, soit par simulation Monte-Carlo.

Pour conclure cette introduction, mentionnons qu'une troisième méthode, dite des arbres d'événements (event trees) est présentée dans ce chapitre. Introduits dans le rapport WASH-1400 [RSS75], ces arbres d'événements n'étaient pas originellement destinés, semble-t-il, à servir de modèles d'analyse quantitative des risques, mais de représentation qualitative de séquences accidentelles. Cependant, très rapidement et essentiellement dans le domaine de la sûreté nucléaire, ils ont été considérés comme des modèles booléens et utilisés comme tels à des fins d'évaluation quantitative.
Les trois premiers paragraphes de ce chapitre sont consacrés respectivement aux diagrammes et réseaux de fiabilité, aux arbres de défaillance et aux arbres d'événements, tandis que leur utilisation en tant que supports de traitements basés sur la simulation de Monte-Carlo fait l'objet du quatrième et dernier paragraphe.

3.1 RESEAUX DE FIABILITE

La méthode du diagramme de fiabilité (aussi appelé diagramme de succès) est historiquement la première méthode utilisée pour l'analyse du fonctionnement des systèmes et donc également de leur défaillance. Elle a l'énorme avantage de donner une représentation fonctionnelle du système étudié, ce qui permet à tous les acteurs de l'étude d'avoir un vision de «terrain».
Les réseaux de fiabilité sont un sur-ensemble des diagrammes de fiabilité. Ils ont les mêmes qualités de représentation que les diagrammes de fiabilité et en étendent le domaine d'application en permettant de faire des études fiabilistes sur tous les systèmes de type réseau (distribution d'eau, d'électricité, ...).
Nous présentons en premier lieu les diagrammes de fiabilité, en raison de leur antériorité.

3.1.1 DIAGRAMMES DE FIABILITE

Les diagrammes de fiabilité permettent de représenter le fonctionnement attendu du système.
Ils sont composés de blocs représentant les composants ou éléments du système. Le terme composant est pris ici dans un sens large. Il correspond aux composants physiques ou à des sous-systèmes. Ceci dépend bien entendu du niveau de détail de l'étude.
La disposition des ces blocs-composants dépend en fait de la fonction assurée par chacun d'eux au sein du système et en interaction avec les autres, et dépend également du type de défaillance considéré. C'est la raison pour laquelle, le diagramme de fiabilité d'un système peut différer de son schéma structurel. Pour illustrer ce propos, considérons un ensemble constitué de deux accumulateurs, délivrant chacun une tension de 12V, disposés physiquement en parallèle à des fins de redondance. L'ensemble fournit une tension de 12V. Si la défaillance considérée pour les accumulateurs est le court-circuit, le diagramme fonctionnel sera un diagramme série, alors qu'il sera un diagramme parallèle en cas de défaillance par circuit-ouvert.
Le diagramme de fiabilité sur la Figure 3.1 correspond à un système qui ne remplit plus sa fonction si le composant A est en panne ou si les composants B et C sont en panne.
Un diagramme de fiabilité peut être lu comme un circuit électrique unidirectionnel composé d'un générateur (l'entrée ou le nœud source), d'un récepteur (le sortie ou le nœud destination ou cible) et d'un ensemble

d'interrupteurs, le courant se propageant du générateur au récepteur. Les blocs interrupteurs ne laissent pas passer le signal lorsqu'ils ne remplissent plus leur fonction. Le système est en panne si le récepteur n'est plus alimenté.

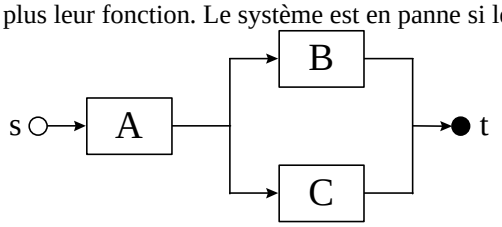
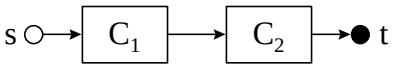
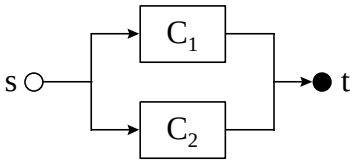
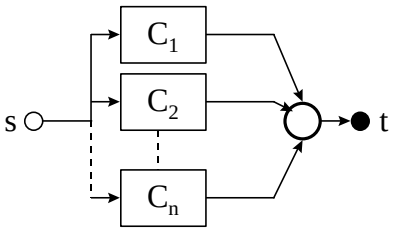
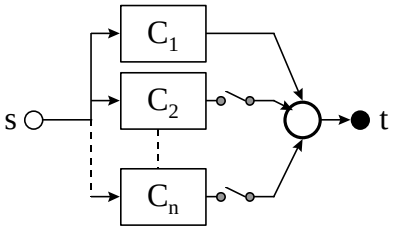


Figure 3.1 : Exemple de diagramme de fiabilité série - parallèle

L'arc entre A et B indique que le bon fonctionnement de B dépend du bon fonctionnement de A. Il représente une relation fonctionnelle entre ces deux blocs élémentaires.

Différentes architectures sont utilisées pour la construction des diagrammes de fiabilité. Elles sont représentées dans le tableau suivant :

Types d'architectures	Représentations graphiques
Série représente le cas où il faut que les deux composants fonctionnent pour que le système fonctionne	
Redondance active ou en parallèle représente le cas où le système fonctionne si au moins un des deux composants fonctionne.	
Redondance active k/n représente le cas où il faut qu'au moins k composants parmi les n fonctionnent simultanément pour que le système fonctionne. Il peut être décrit à l'aide de combinaisons d'architectures en série et en parallèle.	
Redondance passive d'ordre k représente le cas où il existe des composants en réserve qui suppléent successivement le composant actif en cas de panne de ce dernier. Le système de commutation est supposé fiable.	

Il est possible de mesurer des grandeurs fiabilistes directement sur ces diagrammes sans passer par une transformation en formule booléenne.
Lorsque le système n'est composé que d'architectures figurant au tableau ci-dessus et uniquement de composants indépendants et non réparables, il est relativement aisé de calculer la fiabilité (ou disponibilité) du système de manière analytique (Cf. nombreuses monographies sur le sujet, par exemple [AM93], [Vil88]ou [PG80]).

Lorsque l'architecture du système ne relève pas directement d'une des configuration précédentes, il convient, pour s'y ramener, d'utiliser la décomposition de Shannon[2]. Le système de la Figure 3.2 a une architecture plus compliquée à cause du composant C_2. La décomposition de Shannon par rapport à C_2 simplifie le système en architecture parallèle lorsque C_2 fonctionne et en architecture parallèle-série dans le cas opposé.

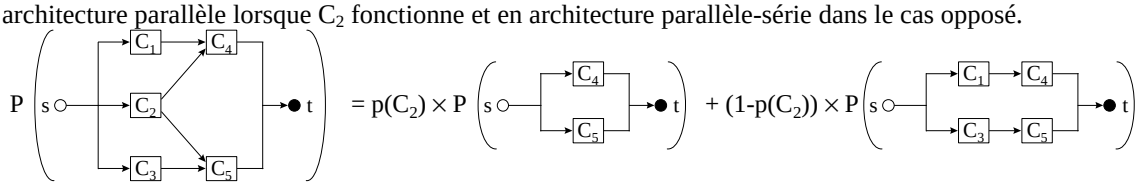


Figure 3.2 : Exemple de résolution d'architecture "complexe"

La Figure 3.3 présente le système précédent sous la forme de deux graphes duaux.

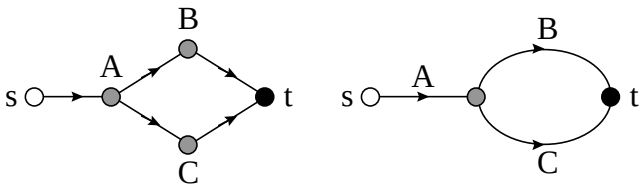


Figure 3.3 : Représentation d'un diagramme de fiabilité sous la forme d'un graphe

La première représentation [Vil88] suppose que les diagrammes de fiabilité sont des graphes admettant une entrée et une sortie dont les sommets (appelés ici blocs) représentent des composants du système et dont les arcs traduisent les relations entre ces différents composants.
La seconde représentation [AM93] considère les nœuds du graphe comme des points de jonction entre les composants du système. Ces derniers sont alors représentés par les arcs du graphe.
Dans tous les cas, nous considérons que les relations entre les composants (les arcs dans le premier cas ou les nœuds dans le second), ainsi que les nœuds source et cible ne sont pas sujets à défaillance.

Ceci nous amène directement à une extension des diagrammes de fiabilité : les réseaux de fiabilité.

3.1.2 RESEAUX DE FIABILITE

Un réseau est un graphe composé d'un ensemble de sommets (ou nœuds) , d'un ensemble d'arêtes et d'une fonction d'incidence reliant chaque arête à deux sommets du réseau. Deux sommets ont des propriétés particulières, car ils correspondent respectivement à la source, notée **s**, et à la destination, notée **t**, du réseau. Les arêtes sont par définition bidirectionnelles. Lorsque l'on voudra préciser une relation unidirectionnelle entre deux sommets, on utilisera la terminologie arc (au lieu de arête) comme cela est de rigueur en théorie des graphes. Une arête entre les nœuds *i* et *j* peut être vue comme deux arcs de sens opposé entre les nœuds *i* et *j*.
Un chemin entre deux sommets A et B du réseau est une succession de sommets et d'arcs reliés entre eux, partant du sommet A et arrivant au sommet B. Il peut y avoir de nombreux chemins entre deux sommets.

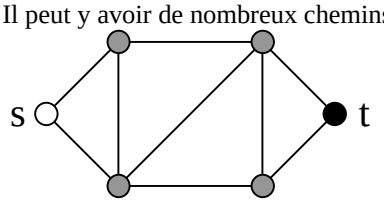


Figure 3.4 : Exemple de réseau de fiabilité

Chaque sommet et chaque arête est soit sujet à défaillance, soit parfaitement fiable.
On appelle chemin ou chemin de succès, tout ensemble de composants (nœuds et arcs) dont le fonctionnement garantit que le but du réseau est lié à la source. Un chemin est minimal si on ne peut lui retirer un composant sans déconnecter la source du but. La notion de chemin minimal correspond donc à l'idée intuitive de chemin de **s** à **t**. L'ensemble de tous les chemins minimaux du système permet l'étude qualitative du fonctionnement du système.
A contrario, l'étude qualitative du dysfonctionnement du système se fait en considérant l'ensemble des coupes minimales du réseau. Une coupe minimale (*minimal cuts set*) correspond au plus petit ensemble de composants dont la panne déconnecte **s** de **t**.

[2] Il est à noter que cette décomposition n'est autre que l'expression du théorème des probabilités totales.

Le système est en état de marche lorsqu'il existe au moins un chemin de succès entre la source et le but du réseau. Quand il n'existe plus aucun chemin de succès entre le nœud de départ et celui d'arrivée, le système est défaillant. Dans ce cas, au moins une coupe minimale est satisfaite.

Un exemple de réseau de fiabilité pourrait être celui d'un réseau téléphonique filaire composé d'un certain nombre de centraux gérant les "aiguillages" et de câbles téléphoniques reliant les centraux entre eux ainsi que les centraux et les particuliers. Les centraux seront représentés par les sommets du réseaux de fiabilité et les connections entre centraux par les arêtes bidirectionnelles.
Par exemple, imaginons qu'une personne (source) téléphone à un membre de sa famille (destinataire) peu après la tempête de décembre 1999. Quelle est la probabilité qu'il y parvienne en supposant qu'il y a une chance sur trois qu'un arbre soit tombé sur chaque câble reliant deux centraux téléphoniques et une chance sur deux que chaque central fonctionne encore (défaillance directe, ou indirecte lorsqu'il n'y a plus d'électricité) ? Existe-t-il encore un chemin de succès entre la source et la cible ?

Notons que les redondances passives et actives d'ordre k des diagrammes de fiabilité sont assez mal représentées dans les réseaux de fiabilité. Dans [Réséda01, DRS96], les redondances actives d'ordre k sont décrites à l'aide d'arcs sûrs directionnels à n entrées et 1 sortie.

3.1.3 COMPILATION VERS LES FORMULES BOOLEENNES

La transformation d'un diagramme de fiabilité en formule booléenne est relativement aisée et intuitive. Toujours par analogie avec un réseau électrique, on considérera que le système fonctionne si le récepteur est alimenté. Or le récepteur (ou d'une manière plus général, un nœud) est alimenté si au moins un des arcs y parvenant est alimenté. Un arc est alimenté s'il n'est pas défaillant et si son nœud amont est alimenté.
Il est toujours possible de faire cette transformation manuellement. Il convient alors de faire attention aux boucles (ou cycles) qui peuvent exister dans un réseau de fiabilité. La règle de construction disant qu'une cause ne peut être postérieure à sa conséquence permet dans tous les cas de déboucler les équations.
Si cette règle peut être mise en œuvre lors de la construction manuelle de la formule booléenne, on lui préférera une transformation automatique.
Il existe deux types d'algorithmes : l'un, proposé par Madre et Coudert dans [MCFB94] (voir aussi [DRS96]), s'applique dans le cas général des réseaux de fiabilité, c'est-à-dire des réseaux de fiabilité incluant les boucles (Notons qu'une arête entre deux nœuds correspond à un cycle). Le principe de cet algorithme est présenté dans l'annexe 1 de ce document. Dans ce cas, la formule booléenne générée utilise un quantificateur existentiel ou universel. Le deuxième algorithme fonctionne par induction arrière (backward induction). Il ne s'applique qu'aux réseaux de fiabilité sans cycle, comme par exemple les diagrammes de fiabilité. Il est détaillé dans la suite de ce chapitre.

Dans la mesure où la modélisation mène à un graphe orienté acyclique, la transformation en équations booléennes peut se faire comme décrit ci-après.
Une manière de représenter le diagramme de fiabilité est d'utiliser deux variables booléennes pour chaque composant x (nœud n_i ou arête a_i). La première, *panne(x)*, est "à vrai" lorsque le composant est en panne. Elle correspond donc à la survenue de la défaillance du composant. La seconde, *fct(x)*, est "à vrai" si et seulement si le composant est alimenté.
Un nœud n_i est alimenté ($fct(n_i)$ est satisfaite) s'il n'est pas défaillant ($panne(n_i)$ est à "faux") et si au moins un des arcs a_k (k de 1 à n) y parvenant est alimenté. En d'autres termes :

$$fct(n_i) = (\neg panne(n_i)) \wedge (fct(a_0) \vee ... \vee fct(a_n)) \qquad [3\text{-}1]$$

Un arc a_j est alimenté ($fct(a_j)$ est satisfaite) s'il n'est pas défaillant ($panne(a_j)$ est vraie) et si son nœud amont k est alimenté. En d'autres termes :

$$fct(a_j) = (\neg panne(a_j)) \wedge fct(n_k) \qquad [3\text{-}2]$$

La formule représentant le bon fonctionnement du système est celle associée au fait que la cible du diagramme est alimentée.

Le diagramme de fiabilité de la Figure 3.5 est représenté sous la forme d'un graphe où les défaillances sont affectées aux nœuds. Les arcs sont sûrs ($g_j = 1 \wedge f_k = f_k$).

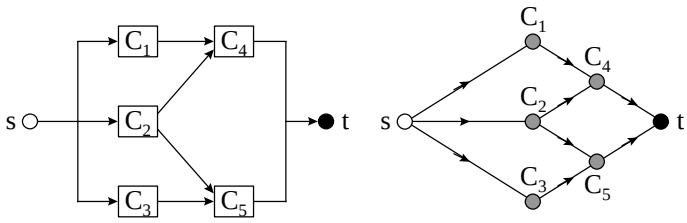


Figure 3.5 : Diagramme de fiabilité

Nous obtenons alors les formules suivantes :

- $fct(s) = (\neg panne(s))$
- $fct(c_1) = (\neg panne(c_1)) \wedge fct(s)$
- $fct(c_2) = (\neg panne(c_2)) \wedge fct(s)$
- $fct(c_3) = (\neg panne(c_3)) \wedge fct(s)$
- $fct(c_4) = (\neg panne(c_4)) \wedge (fct(c_1) \vee fct(c_2))$
- $fct(c_5) = (\neg panne(c_5)) \wedge (fct(c_2) \vee fct(c_3))$
- $fct(t) = (\neg panne(t)) \wedge (fct(c_4) \vee fct(c_5))$

En pratique, les nœuds source et cible sont rarement considérés comme des composants pouvant avoir une défaillance. Dans ce cas, $panne(s) = panne(t) = 0$, d'où $fct(s) = 1$ et $fct(t) = (fct(c_4) \vee fct(c_5))$
Le système fonctionne si le nœud **t** est alimenté, donc si la formule booléenne $path = fct(t)$ est satisfaite.
La formule booléenne *path* code l'ensemble des chemins de succès du réseau.

Au lieu de s'intéresser au bon fonctionnement du système, il est également possible de traduire le dysfonctionnement du système en formule booléenne. Ce problème est le dual du précédent.
La formule (*cuts*) codant l'ensemble des coupes minimales du système est simplement la négation de la formule codant les chemins de succès du système (*path*).

$$cuts = \neg path \qquad [3\text{-}3]$$

Dans notre exemple (Figure 3.5), nous obtenons les formules suivantes en utilisant les lois de De Morgan et en nommant *not-fct(x)* la variable booléenne qui correspondent à la non alimentation du composant x :

- $cuts = \neg fct(t) = panne(t) \vee (not\text{-}fct(c_4) \wedge not\text{-}fct(c_5))$
- $not\text{-}fct(c_5) = panne(c_5) \vee (not\text{-}fct(c_2) \wedge not\text{-}fct(c_3))$
- $not\text{-}fct(c_4) = panne(c_4) \vee (not\text{-}fct(c_1) \wedge not\text{-}fct(c_2))$
- $not\text{-}fct(c_3) = panne(c_3) \vee not\text{-}fct(s)$
- $not\text{-}fct(c_2) = panne(c_2) \vee not\text{-}fct(s)$
- $not\text{-}fct(c_1) = panne(c_1) \vee not\text{-}fct(s)$
- $not\text{-}fct(s) = panne(s)$

3.2 ARBRE DE DEFAILLANCE

Il y a deux approches pour analyser les relations de cause à effet entre les défaillances des composants et la défaillance du système : l'approche inductive et l'approche déductive. Dans une approche inductive, le fiabiliste part d'un ensemble de défaillances élémentaires prises une à une et essaye d'identifier leurs conséquences. C'est une approche du type "Que ce passe-t-il si ...". La méthode des arbres de défaillance est un exemple d'approche déductive, du type "Quelle peut être la cause de ...". Le fiabiliste part, cette fois-ci, d'un événement redouté qui est, en général, une défaillance aux conséquences catastrophiques ou plus simplement une indisponibilité d'un sous-système pouvant affecter la sécurité ou la disponibilité d'un système. Il essaye de déterminer les causes de survenue de cet événement redouté. Ces causes sont en général des événements intermédiaires qu'il faut expliciter par la suite. Le processus déductif est donc poursuivi jusqu'à l'obtention d'événements de base, indépendants entre eux et dont la probabilité d'occurrence est connue.
L'arbre de défaillance représente alors les combinaisons d'événements de base qui conduisent à la réalisation de l'événement redouté.

La méthode des arbres de défaillance (aussi appelé Arbre des Causes, Arbre des Fautes ou encore Arbre des Défauts) a été introduite par Watson, en 1961, au sein de la société Bell Telecom, afin d'évaluer et d'améliorer le système de lancement de missile "Minuteman" au profit de l'US Air Force.

Utilisée dans un premier temps uniquement comme outil de représentation des défaillances des systèmes, cette méthode n'a cessé d'évoluer, aussi bien d'un point de vue méthodologie de construction, que d'un point de vue traitement qualitatif et quantitatif du modèle ainsi obtenu.

3.2.1 REPRESENTATION GRAPHIQUE

La méthode des arbres de défaillance est avant tout un outil graphique qui permet de visualiser clairement les causes d'un événement donné à l'aide d'une structure arborescente. Différents symboles graphiques sont utilisés pour la représentation des événements de l'arbre (Cf. Tableau 3-1).

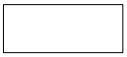
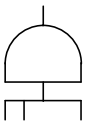
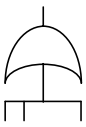
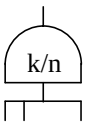
Symbole graphique	Signification du symbole
	Evénement intermédiaire Les causes de cet événement sont développées.
	Evénement de base élémentaire Ne nécessite pas de futur développement. Exemple : Défaillance première d'un composant
	Evénement de base non élémentaire Ne peut être considéré comme élémentaire, mais ses causes ne sont pas et ne seront pas développées.
	Evénement à développer Ne peut être considéré comme élémentaire. Ses causes ne sont pas développées, mais le seront ultérieurement.
	Evénement "Maison" Survenant normalement pendant le fonctionnement du système.
	Evénement "Condition" Utilisé avec certaines portes afin de préciser la condition à satisfaire pour que l'opération logique réalisée par chacune de ces portes s'effectue.

Tableau 3-1 : Représentation des événements

La structure arborescente est réalisée à l'aide de portes (ou opérateurs). Chaque type de porte représente la relation de causalité entre les événements d'entrée de la porte et son événement de sortie. Par exemple, tous les événements d'entrée d'une porte "Et" doivent être réalisés pour que son événement de sortie soit réalisé. Les portes les plus représentatives sont décrites au Tableau 3-2. Les portes fondamentales sont la porte "Et", la porte "Ou" et la porte "K-sur-N".

Symbole graphique	**Signification du symbole**
	Porte "Et" L'événement de sortie est réalisé si toutes les événements d'entrée sont réalisés.
	Porte "Ou" L'événement de sortie est réalisé si au moins un des événements d'entrée est réalisé.
	Porte "K-sur-N" (ou porte combinaison) L'événement de sortie est réalisé si au moins K des N événements d'entrée sont réalisés.

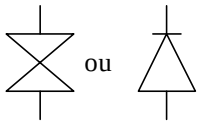
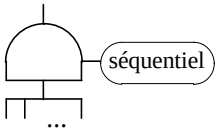
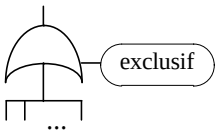
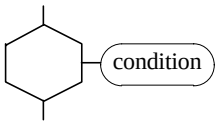
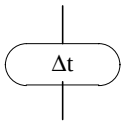
Symbole graphique	Signification du symbole
	Porte "Non" L'événement de sortie est réalisé si l'événement d'entrée ne l'est pas.
	Porte "Et avec condition" L'événement de sortie est réalisé si toutes les événements d'entrée sont réalisés et si la condition est réalisée. Ici, il faut aussi que les événements d'entrée apparaissent séquentiellement.
	Porte "Ou avec condition" L'événement de sortie est réalisé si au moins un des événements d'entrée est réalisé et si la condition est réalisée. Ici, l'événement de sortie est réalisé si un et un seul des événements d'entrée est réalisé.
	Porte "Si" L'événement de sortie est réalisé si l'événement d'entrée est réalisé et si la condition est réalisée.
	Porte "Délai" L'événement de sortie est réalisé si l'événement d'entrée est réalisé depuis et pendant Δt.

Tableau 3-2 : Représentation des portes

Il existe dans la littérature de nombreuses autres portes utilisés dans des applications particulières. Les portes "Quantification", "Sommation" et "Comparaison" [Vil88] sont utilisées afin d'évaluer les pertes d'exploitation sur des lignes de production. Les portes "Redondance passive" et "Séquentielle généralisée" [DSC00] servent dans la description d'arbres de défaillance dynamiques.

Pour finir, il existe des symboles appelés renvois de sous-arbres (aussi appelé transferts de sous-arbres) qui sont utilisés pour éviter de répéter les sous-arbres identiques ou semblables.

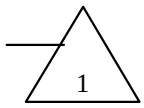
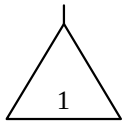
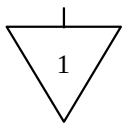
Symbole graphique	Signification du symbole
	Identificateur de renvoi Le sous-arbre commençant par l'événement d'entrée de l'identificateur est transféré aux emplacements signalés par le ou les renvois associés à l'identificateur. Plusieurs renvois peuvent être en liaison avec le même identificateur.
	Renvoi identique La partie de l'arbre qui devrait suivre est identique à celle définie après l'identificateur de renvoi associé.
	Renvoi semblable La partie de l'arbre qui devrait suivre est semblable à celle définie après l'identificateur de renvoi associé.

Tableau 3-3 : Représentation des transferts ou renvois

Un renvoi semblable sert à décrire succinctement un sous-arbre ayant la même structure que le sous-arbre de référence, mais qui représente un système différent. Par exemple, le système de navigation d'un avion de ligne peut comporter deux pilotes automatiques. Ils peuvent être rigoureusement similaires, mais ils seront physiquement différents. La défaillance d'un des composants peut provoquer une défaillance d'un des systèmes de pilotage automatique, mais pas de l'autre.
L'utilisation des renvois semblables permet donc de schématiser rapidement une installation redondante.

3.2.2 METHODOLOGIE DE CONSTRUCTION

Outre l'aspect visuel, le second objectif de la méthode est de déterminer les différentes combinaisons d'événements de base qui entraînent la réalisation d'un événement redouté.

L'événement redouté doit être défini le plus clairement possible. Il ne doit pas être trop général, ni trop spécifique. Une analyse préliminaire des risques aide généralement à identifier le ou les événements redoutés à prendre en compte pour le système étudié. L'événement redouté constituera l'événement de tête de l'arbre de défaillance, c'est-à-dire l'événement-sommet de l'analyse.
Une fois l'événement redouté défini, il convient de préciser les limites de l'étude comme par exemple, la frontière entre le système et son environnement. Dans une étude de disponibilité d'une unité de production B alimentée par une autre unité de production A, on ne recherche pas en général les causes de défaillance de l'unité A entraînant l'arrêt de l'unité B, car elle fait partie de l'environnement de l'unité B. De la même manière les hypothèses de l'étude comme l'état initial, la durée de mission ou les hypothèses simplificatrices du système étudié doivent être clairement définies. Dans le cas de systèmes multi-phases comme ceux de l'aéronautique (décollage, vol, atterrissage) , il convient de préciser la phase étudiée.
Le niveau de détail ou de résolution de l'étude est aussi à définir. Par exemple, est-il nécessaire d'étendre l'analyse jusqu'aux composants, ou suffit-il de s'arrêter au niveau des sous-systèmes pour satisfaire aux objectifs de l'étude ? Ainsi, lors de l'analyse d'un système hydraulique, doit-on s'arrêter au niveau des pompes, vannes et autres constituants, ou faut-il poursuivre l'analyse jusqu'aux tiges, boulons et autres visseries de ces composants ? Le choix du niveau de détail dépend bien entendu des objectifs de l'étude, de l'avancement de la conception du système et du degré de connaissance que l'on a sur ces composants. En particulier, si l'un des objectifs de l'étude est l'obtention de résultats quantitatifs, alors le niveau de détail de l'analyse est fixé par les données de sûreté de fonctionnement dont on peut disposer.

A partir de l'événement redouté, l'arbre de défaillance est développé en déterminant les causes immédiates, nécessaires et suffisantes de son occurrence. Il est important de préciser que ce ne sont pas les causes au niveau des composants qui sont recherchées, mais bien les causes immédiates au niveau de l'événement. Les causes immédiates de survenue de cet événement permettent de le décomposer en événements intermédiaires. Des opérateurs, usuellement des portes logiques (Cf. Tableau 3-2), permettent de décrire les relations entre l'événement - sommet et ses causes - événements intermédiaires. Les événements intermédiaires sont, à leur tour, décomposés et ainsi de suite jusqu'au niveau de décomposition recherché.

D'autres règles ont été élaborées afin de parfaire le processus de construction des arbres de défaillance. Nous n'en préciserons ici que deux :

1. "Pas de miracle à espérer" :
   Dans un système réel, la défaillance d'un composant peut neutraliser miraculeusement la défaillance d'un autre composant. Dans ce cas, le système étudié ne possède pas la propriété de cohérence. Dans l'analyse du système, il faudra supposer que le second composant fonctionne normalement pendant l'existence de la première défaillance.
2. "Les causes sont antérieures aux conséquences" :
   Rechercher les causes d'un événement revient à remonter dans le temps. Les causes d'un événement ne peuvent être postérieures à l'existence de celui-ci. Cette règle facilite l'élaboration d'un arbre de défaillance pour des systèmes bouclés. Un événement ne peut être à la fois cause et conséquence d'un autre événement. Lorsque une des causes de l'événement en cours est une de ses conséquences, il ne faut pas la considérer comme une cause possible.

Il est à noter que cette procédure de construction manuelle demeure délicate dans la mesure où elle relève plus de la maîtrise individuelle de l'analyste que d'une démarche standardisée. Cette constatation a poussé de nombreuses équipes à chercher des solutions alternatives. Ces solutions sont basées en général sur une description (ou analyse) fonctionnelle, puis dysfonctionnelle du système considéré. L'arbre de défaillance est alors déduit automatiquement soit manuellement [DCSB95], soit à l'aide d'outils informatiques [ARB89, GH01].

3.2.3 ELABORATION D'UN ARBRE DE DEFAILLANCE

Considérons le système suivant.

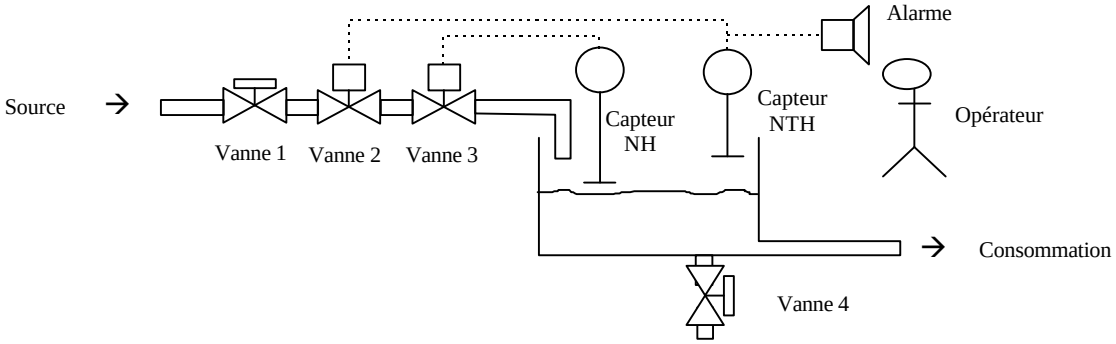


Figure 3.6 : Schéma simplifié d'une installation hydraulique

L'événement redouté est le débordement du réservoir.
En temps ordinaire, la quantité d'eau diminue en fonction de la consommation et augmente en fonction du débit de la source. Si la consommation s'arrête, le niveau augmente jusqu'à ce que le capteur de niveau NH (Niveau Haut) arrête automatiquement l'alimentation en eau en commandant la fermeture de la vanne 3. Ce système est d'une certaine manière redondant. En effet, un second capteur NTH (Niveau Très Haut) permet la fermeture de la vanne 2, si le niveau continue à progresser. De plus, ce capteur prévient un opérateur à l'aide d'une alarme. Ce dernier doit, en cas de défaillance des entités précédentes, fermer manuellement la vanne 1. La vanne 4 d'évacuation est ouverte en dernier recours par l'opérateur. Son débit est supérieur à celui des trois autres vannes et permet de transférer le contenu du réservoir dans un bac de rétention de grandes dimensions, ce qui permet à l'opérateur de procéder aux réparations nécessaires.
De plus, en vertu de la règle "pas de miracle à espérer", on ne peut envisager qu'une demande accrue en eau de la part des consommateurs puisse faire éviter le débordement.

La démarche consiste, à partir de l'événement redouté, d'en expliciter les causes et ceci de manière récursive. L'arbre de défaillance correspondant au débordement de la cuve est schématisé Figure 3.6. Les numéros situés à côté des événements intermédiaires correspondent à l'ordre récursif de l'analyse.

1. Le réservoir déborde si et seulement si il n'y a pas d'arrêt de l'alimentation en eau et si il n'y a pas d'évacuation. Les causes évoquées sont les causes immédiates et suffisantes du débordement du réservoir. Il faut que les deux causes surviennent pour que le débordement du réservoir soit effectif.
   Dans cette étude, nous considérerons que la pluie tombant dans le réservoir ne peut pas provoquer son débordement.
2. L'alimentation du réservoir peut être arrêtée par les vannes 1, 2 ou 3.
3. Il n'y a pas d'arrêt de l'alimentation par la vanne 1, si la vanne 1 est défaillante (bloquée ouverte) ou si elle n'a pas été actionnée.
4. La vanne 1 n'est pas actionnée si l'opérateur est absent ou si l'opérateur n'a pas été averti.
5. L'opérateur n'est pas averti si l'alarme est en panne ou si le capteur NTH commandant l'alarme est défaillant.
6. Il n'y a pas d'arrêt par la vanne 2 (respectivement vanne 3) , si elle est défaillante (Bloquée ouverte) ou si elle n'est pas actionnée.
7. La vanne 2 (respectivement vanne 3) n'est pas actionnée, si elle n'a pas reçu l'information de se fermer, c'est-à-dire si le capteur de niveau NTH (respectivement capteur de niveau NH) est défaillant.
8. Il n'y a pas d'évacuation du réservoir si la vanne 4 reste fermée.
9. La vanne 4 reste fermée, si elle est défaillante (bloquée fermée) ou si elle n'a pas été actionnée.
10. La vanne 4 n'est pas actionnée si l'opérateur est absent ou s'il n'a pas été prévenu. Or cette combinaison d'événements intermédiaires a déjà été explicitée en 4. Nous pouvons donc utiliser un renvoi identique afin d'alléger l'arbre.

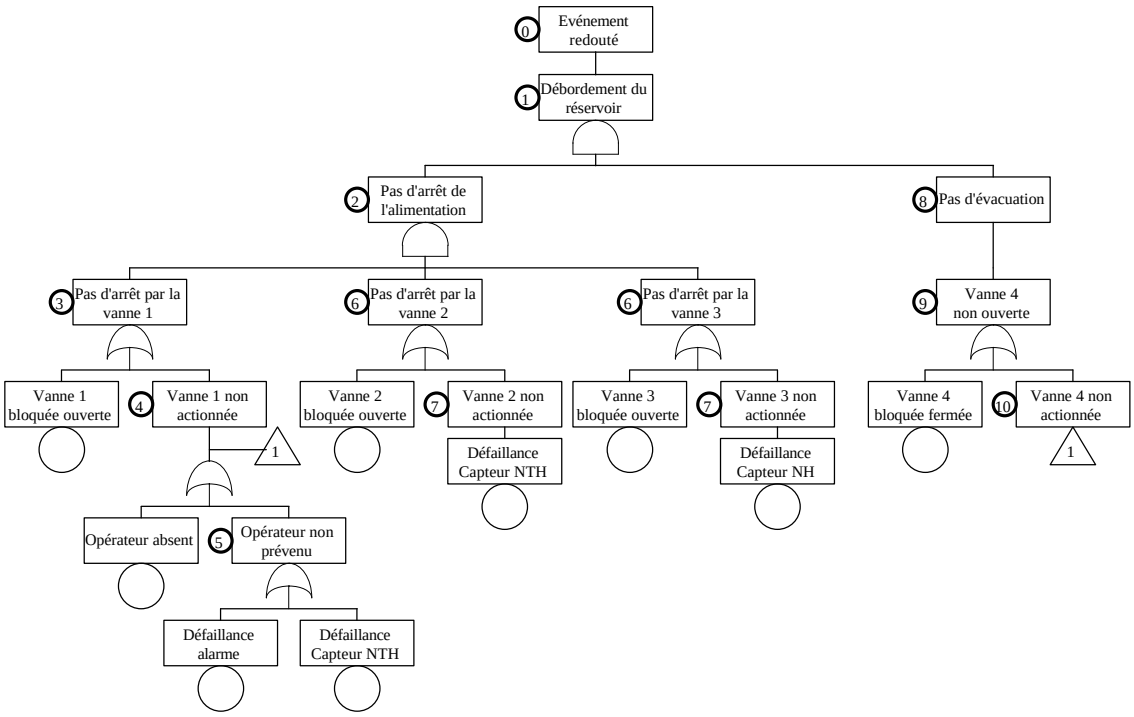


Figure 3.7 : Arbre de défaillance

L'utilisation de renvois identiques lors d'une étude est quelquefois critiqué. En effet, sur des systèmes bouclés, il convient, avant d'utiliser un renvoi identique, de procéder aux deux vérifications suivantes :

- la règle 2 ("les causes sont antérieures aux conséquences") n'a pas été utilisée dans le développement de la référence,
- aucune des causes de la référence n'est une conséquence de l'événement intermédiaire en cours.

3.2.4 TECHNIQUE DES BARRIERES

La technique des barrières est une méthode spécifiquement employée lors de l'exploitation qualitative des arbres de défaillance. Elle est principalement utilisée lorsque les données probabilistes des événements de base ne sont pas connues ou difficilement estimables. Elle a été définie et formalisée pour des études de sûreté, principalement dans le domaine du nucléaire, afin de démontrer que la maîtrise du risque est effective.
Lors d'une étude de sécurité, les événements redoutés sont hiérarchisés en fonction de la gravité des conséquences qu'ils peuvent entraîner. Plus la gravité des conséquences attachées à un événement redouté est importante, plus le nombre de barrières de sécurité à mettre en œuvre doit être important, réduisant ainsi sa fréquence.
Dans le milieu industriel, la diminution de la fréquence de l'événement redouté (mesure de prévention) est préférée à la réduction de la gravité de ces conséquences (mesure de protection). Elle est en général aussi plus facile à mettre en œuvre et moins onéreuse.

Une barrière est une protection censée prévenir l'occurrence de l'événement redouté. Il existe deux types de barrières : les barrières techniques (disjoncteur, alarme, ...) et les barrières d'utilisation (procédures de maintenance, d'exploitation, consignes de sécurité, formation des opérateurs, ...). Les barrières techniques sont en général préférées aux barrières d'utilisation, car elles garantissent *a priori* une meilleure prévention, notamment dans la durée.

La mise en œuvre de la méthode a pour objectif de garantir l'existence et le maintien dans le temps d'un nombre suffisant de barrières prévenant l'occurrence de l'événement redouté. Le nombre de barrières est généralement défini d'une manière globale au niveau de l'installation, en fonction de la gravité des conséquences pouvant être générées par l'apparition de l'événement redouté. On commence par définir une échelle de gravité. Celle-ci associe à chaque niveau de gravité un nombre de barrières parmi lesquelles figurent des barrières techniques dont le nombre peut être spécifié.
Il est important de noter qu'il convient de garantir de l'intégrité de ces barrières dans le temps.

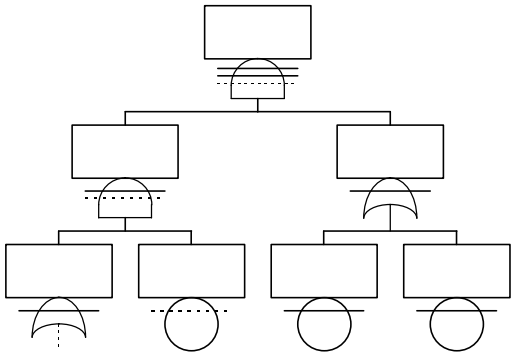


Figure 3.8 : Exemple d'utilisation de la technique des barrières

Le nombre de barrières à mettre en œuvre peut être vu comme une contrainte sur un événement donné. Suivant le type de dépendance entre l'événement courant et ses événements-causes, la contrainte est distribuée différemment. Une porte "Ou" transmet à chacun de ses événements d'entrée la contrainte de son événement de sortie. Une porte "Et" répartit, en fonction de critères techniques et économiques, la contrainte sur l'ensemble de ses événements d'entrée. Une répartition possible est de tout affecter à un seul événement d'entrée.

L'exemple de la Figure 3.8 est une représentation de la technique des barrières avec 3 barrières (2 barrières techniques en trait plein et 1 barrière d'utilisation en pointillé).
Cette technique qualitative ne s'applique que sur des arbres de défaillance cohérents, issus en général d'une analyse manuelle.

La technique des barrières n'est rien d'autre qu'une adaptation aux arbres de défaillance de la méthode des barrières utilisée dans l'ingénierie nucléaire dans les années 60-70. Cette dernière consiste à protéger le public contre une conséquence d'un relâchement accidentel de produits de fission en interposant une suite de barrières "étanches". On examine chacune des barrières (en général au nombre de 3 ou 4 : la gaine, l'enveloppe du circuit et le confinement du circuit primaire) en fonction des différentes dégradations qu'elles pourraient subir en fonctionnement normal ou même lors de certains transitoires accidentels.
L'adoption de la filière des réacteurs à eau sous pression ne permettait plus d'utiliser une telle méthode, car dans ce type de réacteurs il n'est pas possible de définir des barrières "étanches" indépendantes les unes des autres. Ceci a conduit les autorités de sûreté à adopter le concept de défense en profondeur.
Ce concept "consiste en une suite d'actions, d'équipements ou de procédures, regroupés en niveaux dont chacun a pour objectif de prévenir les dégradations susceptibles de conduire au niveau suivant et de limiter les conséquences de la défaillance du niveau précédent" [Lib95].
Actuellement la défense en profondeur est structurée généralement en cinq niveaux :

1. Prévention des anomalies de fonctionnement et des défaillances des systèmes
2. Maintien de l'installation dans le domaine autorisé
3. Maîtrise des accidents à l'intérieur des hypothèses de conception
4. Prévention de la dégradation des conditions accidentelles et limitation des conséquences d'accidents graves
5. Limitation des conséquences radiologiques pour les populations en cas de rejets importants

Ces actions sont des démarches qualitatives et déterministes.

En complément de ces études, les études probabilistes de sûreté (EPS) ont comme objectif direct l'estimation de la probabilité de fusion du coeur, de rejets ou d'accidents graves. Ces EPS sont construites pour la plupart à partir des arbres d'événements présentés au chapitre suivant.

3.3 ARBRE D'EVENEMENTS

Les premières utilisations de la méthode des arbres d'événements datent des années 70. Présentée dans le rapport WASH-1400, dit de Rasmussen [RSS75], comme un moyen de représentation synthétique des différentes séquences accidentelles conduisant à divers événements redoutés au sein des centrales nucléaires à eau pressurisée, elle a depuis été également utilisée comme support d'évaluation quantitative de l'occurrence des séquences précitées. Cette extension de son domaine d'utilisation peut s'expliquer aussi bien par ses qualités de représentation, que par obligation vis-à-vis des organismes de tutelle, tel que l'IPSN en France, chargés de contrôler la bonne exécution des études probabilistes de sûreté conduites par les exploitants.

Son utilisation massive dans le nucléaire peut s'expliquer par la particularité des installations mises en œuvre dans ce type d'industrie. Ces dernières sont en général très sécurisées. De nombreuses redondances fonctionnelles et physiques sont présentes. Lorsqu'un événement mettant en danger l'installation survient, il existe de nombreuses parades afin que les conséquences de cet événement n'aboutissent qu'à une baisse de production, à la rigueur à l'arrêt de l'unité de production et au pire à sa perte sans danger réel pour l'environnement. La seule méthode permettant d'étudier un tel système est une étude systématique des défaillances de tous les composants pouvant avoir des conséquences à court, moyen et long terme sur la sûreté de l'installation.
La méthode des arbres d'événements a aussi été utilisée, mais d'une manière marginale, dans d'autres domaines que le nucléaire. Citons l'industrie chimique, le ferroviaire et même la robotique [Kho96].

La suite de ce chapitre est organisée de la manière suivante : Le paragraphe 3.3.1 présente succinctement ce que sont les études probabilistes de sûreté (EPS), pratiquées majoritairement dans le nucléaire. Le paragraphe 3.3.2 introduit la méthode de l'arbre d'événements (AE), tandis que le suivant est consacré à une présentation des différents types d'AE existants. Ceux-ci se différencient par leur capacité à traiter les diverses dépendances pouvant exister entre plusieurs systèmes ou sous-systèmes de sûreté. Le dernier chapitre traite de deux démarches de modélisation des AE, relativement opposées.

3.3.1 ETUDES PROBABILISTES DE SURETE

Une EPS d'un réacteur nucléaire a pour objet, rappelons-le, d'identifier tous les scénarios d'accident susceptibles de se produire en endommageant gravement le réacteur et d'en évaluer les fréquences d'occurrence. Elle peut permettre, par là-même, de juger de la pertinence des mesures de prévention et de protection prévues ou mises en place et conduire à d'éventuelles améliorations de la sûreté, tant au niveau de la conception que de l'exploitation d'une installation nucléaire.
On pourra consulter avec profit la référence [Lib95] qui expose les principes qui régissent la gestion du risque nucléaire en France et à l'étranger, des années 70 à nos jours.

Les types d'accidents envisagés sur les réacteurs

Dans un réacteur à eau sous pression, le combustible est composé de pastilles d'oxyde d'uranium enrichi en isotope 235 disposées dans des gaines cylindriques en zircaloy, elles-mêmes regroupées en faisceaux formant des éléments combustibles. Ces derniers, qui constituent le cœur du réacteur, sont placés dans une cuve en acier reliée à des générateurs de vapeur, transmettant l'énergie à un circuit secondaire. À l'intérieur du circuit primaire se trouve de l'eau sous pression à 155 bars et 300°C environ. Des grappes de commande contenant des absorbants neutroniques s'insèrent dans le cœur, à partir du haut, de façon à contrôler la réaction nucléaire. Le circuit secondaire produit de la vapeur à 60 bars et 280°C qui est envoyée à la turbine alimentant un alternateur. L'enceinte de confinement en béton entourant le circuit primaire forme une barrière entre les produits radioactifs pouvant se dégager du combustible en cas d'accident et l'extérieur.
La fission de l'uranium et du plutonium donne naissance à des corps radioactifs ou produits de fission qui dégagent de l'énergie, même après l'arrêt de la réaction nucléaire. La puissance résiduelle représente 7 % de la puissance totale après l'arrêt du réacteur et décroît rapidement avec le temps (1 % après quatre heures). Les produits radioactifs restent à l'intérieur du combustible tant que celui-ci est bien refroidi. Pour qu'ils s'échappent, il faudrait que le combustible soit surchauffé ou fondu. Ce phénomène n'a lieu que si la vitesse de production de chaleur dans le combustible est supérieure à celle d'évacuation. Ce type de déséquilibres thermiques peut être provoqué par un événement du type perte de réfrigérant ou par des événements transitoires.
Les événements transitoires incidentels ou accidentels comprennent toutes les situations susceptibles de provoquer des déséquilibres thermiques dans le combustible, réacteur en marche ou à l'arrêt. Ils sont corrigés automatiquement par les systèmes de régulation de la centrale qui ramènent le réacteur dans ses conditions normales de fonctionnement. Des transitoires plus importants exigent l'arrêt de la réaction nucléaire. Le système de protection doublé, secouru électriquement, remplit ce rôle en commandant la chute des grappes de contrôle dans le cœur. L'évacuation de l'énergie du circuit primaire ne pouvant plus être assurée par le circuit secondaire de refroidissement, il est alors nécessaire d'alimenter en eau les générateurs de vapeur avec un système auxiliaire indépendant (ASG). C'est seulement dans le cas où le système de protection ou celui d'alimentation de secours en eau des générateurs de vapeur est défaillant qu'une surchauffe du combustible est à craindre. Dans les études probabilistes, il est également envisagé la perte des dispositifs indispensables au fonctionnement des systèmes de protection et de sauvegarde : les circuits d'alimentation électrique et ceux de refroidissement des composants actifs.
Parmi les événements transitoires accidentels majeurs, on distingue les accidents de perte de réfrigérant primaire. Ils seraient initiés par des défaillances de l'enveloppe du circuit primaire et pourraient conduire à des brèches de diamètre maximum égal à 60 cm. De façon à pallier les conséquences d'une telle défaillance, des systèmes de sauvegarde ont été mis en place. Le système d'injection de sécurité (RIS) permet d'envoyer de l'eau dans le circuit

primaire et d'éviter un échauffement excessif du combustible. Le système d'aspersion dans l'enceinte (EAS) permet d'évacuer la puissance résiduelle hors de l'enceinte de confinement.
Si ces systèmes fonctionnent, les accidents de perte de réfrigérant n'aboutissent, dans les cas les plus graves, qu'à des ruptures limitées des gaines du combustible dont la température ne dépasse pas 1200°C. L'émission de produits de fission dans l'enceinte de confinement est alors faible. L'enceinte étant conçue pour résister à la pression engendrée par la vaporisation de l'eau du circuit primaire, les rejets radioactifs vers l'extérieur à travers les petits défauts d'étanchéité n'entraînent que des conséquences radiologiques très limitées (de l'ordre de 200 GBq en Iode 131 équivalent). C'est seulement dans le cas où l'un des systèmes de sauvegarde n'assurerait pas sa fonction que l'accident pourrait conduire à la fusion du cœur.

L'évaluation de la probabilité de fusion du cœur

L'événement dont on cherche à évaluer la probabilité est rare (citons l'accident de fusion du cœur observé à la centrale à eau sous pression de Three Mile Island pour 2000 années x réacteurs). Le principe consiste à le décomposer en enchaînements d'événements de fréquence observable. La démarche poursuivie est donc de construire tous les scénarios, ou "séquences d'accident", aboutissant à la fusion du cœur. La définition des séquences d'accident se fait, nous l'avons déjà mentionné, à l'aide "d'arbres d'événements". Chaque branche de l'arbre d'événements constitue une séquence d'accident dont on détermine, par une analyse fonctionnelle et physique, si elle conduit ou non à la fusion du cœur.
La méthode envisagée consiste dans un premier temps à recenser tous les événements initiateurs à prendre en compte. Les événements initiateurs sont ceux qui impliquent la mise en œuvre d'au moins un système de sûreté ou de sauvegarde. Ils peuvent être du type "rupture du circuit primaire" ou du type "transitoire d'exploitation". À partir d'un événement initiateur, les systèmes et les procédures de conduite intervenant au cours de l'accident, dont la défaillance est systématiquement envisagée, sont recensés.

3.3.2 PRESENTATION GENERALE DE LA METHODE

Alors que la démarche des arbres de défaillance est de rechercher les causes d'un événement non souhaité, la méthode des arbres d'événements s'intéresse à ses conséquences possibles. Il s'agit donc d'une méthode plutôt inductive.
A partir d'un événement initiateur (défaillance d'un composant, erreur humaine, ...) l'arbre d'événements va donc fournir toutes les évolutions possibles. La survenue de l'événement initiateur va modifier le comportement du système jusqu'à ce que le dépassement d'une valeur-seuil déclenche un processus de sauvegarde du dispositif. Ce processus consiste à prendre des mesures, soit automatiquement, soit par l'intermédiaire d'opérateurs, afin de faire évoluer le système dans un certain domaine de sécurité. Ces mesures peuvent ne pas aboutir pour différentes raisons et, suivant leur succès ou leur échec, le système évoluera de manière différente. Ces mesures de protection sont aussi appelés événements génériques. On représente la réussite ou l'échec de ces mesures à l'aide de branchements dans l'arbre d'événements.

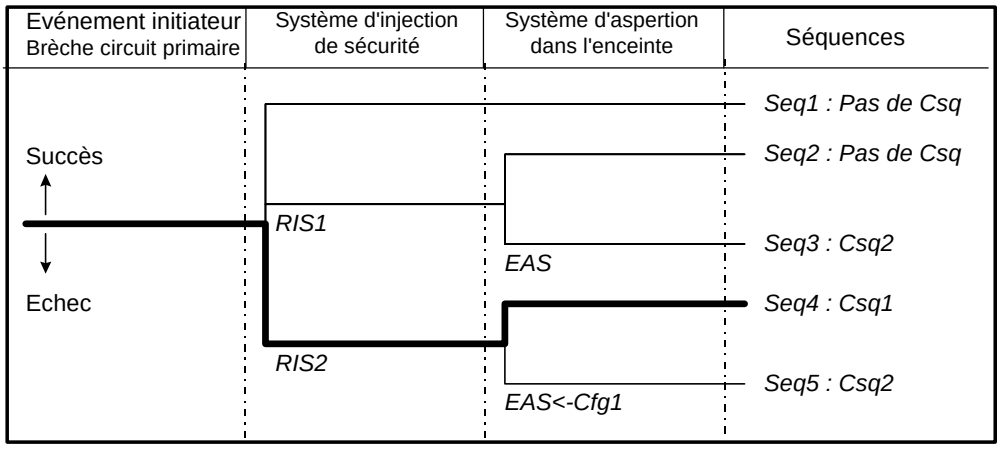


Figure 3.9 : Exemple d'arbre d'événements

La méthode des arbres d'événements se base sur un formalisme graphique représenté à la Figure 3.9. Un arbre d'événements code un ensemble de séquences d'événements. Chaque séquence consiste en un événement initiateur (la colonne la plus à gauche) et une succession de défaillances ou de fonctionnements de systèmes de sûreté. Elle aboutit à un état du système (la colonne la plus à droite). Des conséquences (Csq) que l'on juge acceptables ou non-acceptables sont associées à ces états.

Chaque colonne intermédiaire concerne un système de sûreté donné.
Chaque branchement correspond aux états possibles du système de sûreté. Par convention, la branche supérieure correspond au bon fonctionnement du système. Les autres branches représentent les pannes du système (partielles ou totales). Afin de déterminer dans quelle branche de l'alternative se trouve le système, des événements redoutés sont associés aux branches de non-fonctionnement.

Par exemple, la séquence en trait gras sur la Figure 3.9 correspond à la séquence d'événements suivante : survenue d'une brèche dans le circuit primaire, panne du système d'injection de sécurité (RIS2), succès du système d'aspersion dans l'enceinte (EAS).
Les branchements multiples permettent de considérer d'autres alternatives que le fonctionnement ou la panne totale. Lors de leur utilisation, il convient de vérifier qu'il n'existe pas de non déterminisme. Le système ne peut être que dans un état possible. Il ne peut pas y avoir de configurations qui mettent le système en panne totale et en panne partielle.

La méthode des arbres d'événements comporte un aspect temporel dans la mesure où les systèmes de sécurité sont considérés dans l'ordre chronologique de leur intervention. La première mesure de correction est mise en œuvre lorsque l'événement initiateur est survenu. Puis les systèmes de protection sont sollicités ou non suivant que le système précédent a ou n'a pas rempli sa fonction dans un laps de temps donné.

Il existe des difficultés mathématiques inhérentes à la quantification des arbres d'événements. Elles sont dues à la prise en compte effective des dépendances pouvant exister entre les différents systèmes de sûreté appartenant à une même séquence. Ces dépendances peuvent être de deux ordres :

- Des systèmes comme les générateurs électriques ou les réserves d'eau sont des système pouvant alimenter plusieurs systèmes de sûreté. Ils correspondent donc à des défaillances de cause commune. Si tel est le cas, le calcul de la probabilité d'occurrence d'une séquence doit tenir compte des éventuelles dépendances fonctionnelles entre systèmes de sûreté.
- On peut ensuite imaginer que le système d'aspersion de l'enceinte ne soit déclenché qu'après que le système d'injection de sûreté ait été sollicité sans succès. On peut donc aussi avoir une dépendance temporelle entre les événements. Le problème est alors de quantifier les séquences faisant intervenir le système RIS puis le système EAS, sachant que le premier peut avoir une durée d'intervention variable et que l'on ne connaît donc pas l'instant de déclenchement du second.

Si les événements redoutés représentant les défaillances des systèmes de sûreté sont considérés comme étant indépendants et s'il n'existe pas de dépendance temporelle entre ces événements élémentaires, la probabilité d'un scénario s'obtient simplement en multipliant les probabilités de succès ou d'échec des systèmes de sûreté sollicités au cours de ce scénario.
La somme des probabilités des scénarios conduisant à un état sans conséquence nous fournit alors la probabilité que le déroulement accidentel, initié par l'événement de départ, soit arrêté par au moins un système de sûreté.

Le chapitre qui suit présente différents types d'arbres d'événements qui se différencient par la manière dont ils prennent en compte les événements redoutés associés aux branches de non-fonctionnement.

3.3.3 DIFFERENTS TYPES D'ARBRES D'EVENEMENTS

Arbres d'événements booléens

Il s'agit de l'approche la plus répandue dans les études nucléaire de par le monde. Elle consiste à modéliser la défaillance des systèmes de sûreté à l'aide d'arbres de défaillance. L'équivalent booléen d'un scénario correspond à la conjonction des événements rencontrés le long des branches qui le constituent. En résolvant la formule booléenne dans son intégralité, les dépendances fonctionnelles entre systèmes de sécurité sont bien prises en compte.

Dans ce cadre théorique, la prise en compte des dépendances temporelles entre événements génériques est beaucoup plus difficile à appréhender et se ramène au problème général de la prise en compte de phénomènes temporels dans les arbres de défaillance. Cette approche nous donne une vue statique de notre système. L'arbre d'événements ainsi considéré prend en compte la succession logique et ordonnée des événements possibles, mais il ne considère pas les délais entre ces différents événements. Or l'inertie des systèmes considérés nous suggère qu'il est nécessaire de les prendre en compte. Le temps de réaction d'un opérateur à une situation donnée peut être

déterminant sur l'évolution d'un scénario. Les arbres d'événements continus ou dynamiques présentés par la suite essayent de répondre à ces questions.

De très nombreux logiciels, supports des EPS, utilisent une approches mixte ET/FT (Event Tree / Fault Tree). Malheureusement, ils ne prennent pas forcément en compte les relations de dépendance entre les systèmes de sûreté.
Dans la plupart des cas, il recherche pour chaque système de sûreté la probabilité d'occurrence de l'arbre de défaillance associé et il l'injecte dans l'arbre d'événements en considérant le système de sûreté comme indépendant, ce qui nous ramène à l'approche la plus élémentaire.
Au mieux, seules les dépendances entre les systèmes de sûreté défaillants sont prises en compte. Ils considèrent qu'un système de sûreté fonctionnant correctement n'est pas à prendre en compte, ni dans la logique du scénario, ni dans sa probabilité associée. Si l'approximation probabiliste est recevable (lorsque toutes les défaillances élémentaires ont une très faible probabilité d'occurrence, nous pouvons considérer que la probabilité de bon fonctionnement d'un système de sûreté est très proche de 1 et donc la prendre égale à 1) l'approximation en termes de logique ne l'est pas. Lorsqu'un système de sûreté fonctionne, c'est qu'au moins un certain nombre de composants de ce système fonctionnent. Si ces composants sont utilisés dans d'autres systèmes de sûreté, il n'est pas admissible de les considérer comme potentiellement en panne. Il faut donc bien les prendre en considération d'une manière ou d'une autre.

Cette approche est exposée de manière plus approfondie dans la seconde partie de ce document.

Arbres d'événements multi-phases

Un arbre d'événements peut se définir comme un système multi-phases, chaque phase correspondant au fonctionnement d'un système de sûreté pendant un temps de mission donné. L'aspect chronologique intrinsèque de l'arbre d'événements est représenté par l'enchaînement des phases du système.

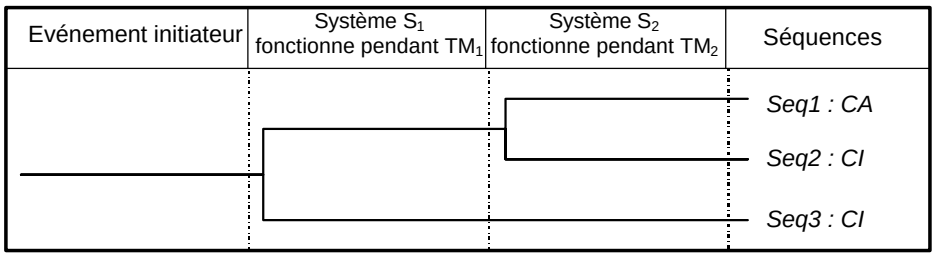


Figure 3.10 : Arbre d'événements de type multi-phases

Dans notre cas, la probabilité d'occurrence associée à la première séquence est égale à la probabilité d'occurrence de l'événement initiateur multipliée par la probabilité que le système multi-phasé S constitué des deux systèmes élémentaires S_1 et S_2 fonctionne sur la durée TM_1+TM_2, c'est-à-dire par la fiabilité du système S à TM_1+TM_2.
Terpstra [Ter84] a montré que le calcul de la fiabilité de systèmes multi-phases (en considérant que chaque phase est décrite par un arbre de défaillance cohérent) permet de prendre en considération les dépendances fonctionnelles entre les différentes phases et les possibles réparations des composants pendant les phases où ils ne sont pas sollicités.

Arbres d'événements avec dépendances temporelles

Il n'existe que peu de dépendances temporelles traitables par des moyens analytiques. Certains modèles permettent tout de même de traiter les types de dépendances suivants :

- La redondance passive : elle correspond au fait qu'un système, jusque là en attente, démarre pour suppléer un autre système qui vient de tomber en panne. Dans l'arbre d'événements de la Figure 3.11, t_1 représente l'instant où le système S_1 tombe en panne. T représente la durée qui permet de réduire de façon satisfaisante la puissance résiduelle du réacteur avant de l'arrêter. Les deux systèmes S1 et S2 maintiennent le refroidissement du réacteur. Lorsque le premier tombe en panne, le second est aussitôt mis en route.

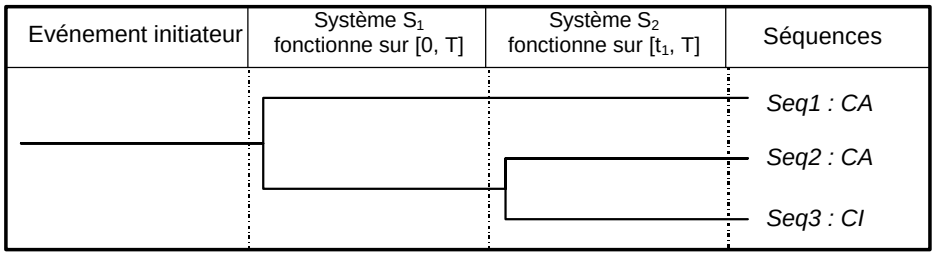


Figure 3.11 : Arbre d'événements avec redondance passive

En prenant en compte un certain nombre d'hypothèses (indépendance fonctionnelle entre les systèmes de sûreté et l'événement initiateur, pas de défaillance au démarrage pour S_1 et S_2, systèmes non réparables, ...), la probabilité de la séquence 3 peut être obtenue en considérant le produit de convolution des densités de défaillance des systèmes S_1 et S_2.

- La récupération d'un initiateur ou d'un système de sûreté : ce type de dépendance se rencontre lors de transitoires réparables. Considérons la survenue d'un événement initiateur. Cet événement initiateur peut disparaître, car il correspond à la panne d'un système réparable. Un ou plusieurs systèmes de sûreté sont mis en route afin de permettre à l'événement initiateur de disparaître sans dégrader la sûreté de l'installation.
  Il est même possible de considérer les systèmes de sûreté comme réparables et de considérer qu'il existe un délai entre la perte du dernier système de sûreté et la survenue des conséquences inacceptables. Ce délai permettant de réparer un des systèmes de sûreté ou de faire disparaître l'événement initiateur.

Le logiciel LESSEPS (et plus particulièrement le cœur de calcul ISA : Intégration de Séquences Accidentelles) de EdF est capable de traiter ce genre de modèle. Dans ce logiciel, la défiabilité liée aux événements redoutés associés aux branches d'échec des arbres d'événements est définie par un polynôme de défiabilité de degré 4. Les systèmes de sûreté et l'initiateur peuvent être réparables (temps de réparation constant). Les systèmes peuvent être en redondance passive. Un délai entre la perte du dernier système et l'atteinte des conditions inacceptables peut être spécifié. En revanche les dépendances fonctionnelles existant entre les systèmes de sûreté ne sont pas considérées.
Le peu de références concernant ce logiciel ou les algorithmes mis en œuvre n'a pas favorisé leur diffusion au-delà des services d'EdF qui les utilisent.

Arbres d'événements flous

La logique floue a été introduite par Zadeh en 1965. Cette science "repose sur une théorie rigoureuse permettant de représenter et de manipuler des connaissances imparfaitement décrites, vagues ou imprécises et d'établir une interface entre des données décrites symboliquement (avec des mots) et numériquement (avec des chiffres)" [Gac97].
L'idée de base est d'utiliser l'algèbre floue afin de rendre compte des incertitudes et des imprécisions des données de base employées dans les arbres d'événements.
Pour cela, la probabilité de défaillance (ou plus généralement le taux de défaillance) des systèmes de sûreté est représentée par une fonction d'appartenance. Une fonction d'appartenance, dont un exemple est représenté Figure 3.12, permet de définir dans quelle mesure un élément appartient à une classe. Il s'agit dans notre cas d'une prise en compte de l'incertitude des données de fiabilité.

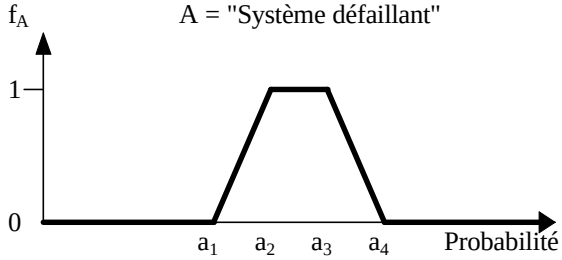


Figure 3.12 : Fonction d'appartenance de type trapèze

La meilleure manière de comprendre ce qu'est une fonction d'appartenance est de se mettre à la place du concepteur ou de l'exploitant d'un système de référence et de répondre à l'affirmation suivante : Votre système a une probabilité de défaillance égale à p. Votre réponse pourrait être du type "Impossible", " Ça m'étonne", "Probable", "Je m'en serais douté". La fonction d'appartenance est nulle si vous considérez qu'une telle probabilité est impossible, mais égale à l'unité lorsque p est égale à l'image que vous vous faites de la probabilité

de votre système, ou encore comprise entre 0 et 1 si l'affirmation précédente vous semble plus ou moins acceptable. La fonction d'appartenance reflète donc le degré de confiance que vous accordez au fait que la probabilité de défaillance de votre système est égale à p.
Une fois les fonctions d'appartenance définies pour tous les systèmes de sûreté de l'arbre d'événements, la théorie des sous-ensembles flous peut être mise en œuvre de manière classique afin de fournir une mesure de possibilité que tel initiateur aboutisse à tel état donné du système ou à telle conséquence.

L'approche des arbres d'événements flous se base sur le constat que l'approche mixte ET/FT est en général utilisée sans considération de dépendances fonctionnelles. L'analyse traditionnelle des arbres d'événements représente la défaillance des systèmes de sûreté par une simple probabilité non entachée d'incertitude.

Les différentes méthodologies proposées dans la littérature ne diffèrent en général que par la manière de définir les fonctions d'appartenance.
Ainsi Kenarangui [Ken91] considère que même l'estimation probabiliste d'un système de sûreté n'est guère raisonnable et qu'il vaut mieux la représenter à l'aide d'une probabilité exprimée en langage naturel (très souvent, souvent, normalement, rarement, très rarement).
Par contre Huang, Chen et Wang [HCW01] proposent de distinguer, parmi les systèmes de sûreté, ceux qui font intervenir des systèmes physiques de ceux qui relèvent d'actions humaines. Dans le premier cas, ils suggèrent de définir la fonction d'appartenance à partir de la moyenne et d'un facteur d'erreur issus d'une étude de sensibilité sur l'arbre de défaillance associé. Alors que des jugements d'expert, reposant sur des variables linguistiques, et une agrégation de ces jugements, permettent de définir la fonction d'appartenance relative aux erreurs humaines.

Conclusion

Parmi les différents types d'arbres d'événements évoqués précédemment, rares sont ceux qui ont été réellement utilisés lors d'études probabilistes de sûreté en contexte opérationnel, même si chacun d'eux a fait l'objet d'applications validant leur emploi (études de faisabilité). En réalité, seuls les arbres booléens sont actuellement acceptés par les autorités de sûreté pour réaliser des EPS. Outre leur antériorité historique, leur facilité d'emploi et les approximations calculatoires qu'ils supportent expliquent ce fait. Il n'en demeure pas moins que ce type d'AE n'est ni idéal, ni même réaliste.
Quant aux approximations mentionnées, si elles ont été validées sur des cas-tests représentatifs, elles n'ont jamais été étudiées dans le cadre d'une EPS complète. Le chapitre 6 présente une telle étude.
Cependant, conscients des imperfections du modèle booléen, certains auteurs ont cherché à l'améliorer par une prise en compte des dépendances fonctionnelles. C'est l'objet du prochain paragraphe.

3.3.4 METHODOLOGIE DE MODELISATION

Dans le cadre d'une approche mixte ET/FT, il existe fondamentalement deux méthodologies de modélisation relativement opposées : La méthode des SET (pour Small Event Tree) aussi appelé "Arbres de défaillance liés" et la méthode des LET (pour Large Event Tree) appelé aussi "Arbres d'événements avec conditions aux limites". La différence principale entre les deux méthodes est que l'une cherche à réduire les dépendances fonctionnelles entre les systèmes de sûreté, tandis que l'autre fait confiance aux logiciels de quantification pour traiter ces dépendances fonctionnelles. Ce n'est pas la première fois que des tentatives d'ordre méthodologique sont faites pour dépasser ou contourner les limites propres à certains modèles ou outils.
La présentation des deux méthodes précédentes (SET/LET), qui suit, s'inspire de celle donnée dans la référence [Ras92].

L'approche S.E.T.

C'est l'approche la plus répandue dans le nucléaire. Elle se base sur le fait qu'aujourd'hui les approximations que l'on peut faire pour résoudre les problèmes de dépendance fonctionnelle sont bonnes et faciles à implémenter. Les approximations utilisées par les logiciels traitant des EPS sont étudiées au chapitre 6.
Les principales étapes de la méthode SET sont décrites ci-après :

1. Pour chaque événement initiateur, un arbre d'événements est construit. Cet arbre d'événements est exprimé en termes de fonctions de sûreté qui sont à leur tour exprimées en termes de systèmes de sûreté.
2. Des arbres de défaillance sont développés pour chaque système de sûreté. Les systèmes supports, comme les alimentations électriques des systèmes de sûreté, sont directement inclus dans les arbres de défaillance. C'est là le point crucial de l'analyse, car les dépendances fonctionnelles sont introduites dans le modèle logique par l'intermédiaire de ces systèmes-supports.
3. Les séquences de l'arbre d'événements s'expriment alors en fonction du succès ou de l'échec de l'intervention des différents systèmes de sûreté successivement sollicités, la prise en compte effective des dépendances fonctionnelles lors de la quantification de ces séquences étant du ressort du logiciel utilisé dans ce but.

L'approche L.E.T.

Cette approche est une autre voie de construction des arbres d'événements. L'idée de base est de structurer les arbres d'événements de telle manière que les événements redoutés associés aux branches de défaillance appartenant à une même séquence soient indépendants les uns des autres. Ceci induit l'obligation de séparer les systèmes supports des systèmes de sûreté les utilisant. Les défaillances de ces deux types de système sont développées sous la forme d'arbres de défaillance et intégrées ainsi à l'arbre d'événements.
L'ordre dans lequel on considère les systèmes au sein de l'arbre d'événements est un point important. Une règle générale est de placer tout système support avant les systèmes l'utilisant.

- Dans ce cas, l'analyse de la défaillance d'un système de sûreté dépend de la branche en cours et donc de la séquence envisagée. En effet, suivant le fonctionnement ou la panne de ces systèmes-supports, le système de sûreté considéré a un fonctionnement plus ou moins dégradé.
- Dans le cas contraire, il n'est pas possible, lors de l'analyse d'un système de sûreté, de faire des suppositions sur l'état de marche ou de panne de ces systèmes supports.

Pour faciliter l'utilisation de tels arbres d'événements, on peut utiliser une matrice de dépendance. Il s'agit d'un tableau dans lequel sont recensés l'ensemble des systèmes de sûreté, ainsi que l'ensemble des systèmes-supports et, d'une manière plus générale, tous les sous-systèmes qui constituent des ressources partagées par plusieurs systèmes. La nature des diverses relations de dépendance est ainsi spécifiée au sein de ce tableau, ce qui permet de distinguer les systèmes de sûreté des systèmes-ressources et de réaliser l'indépendance des divers arbres de défaillance associés au sein d'une même séquence. La probabilité d'occurrence d'une séquence est alors directement obtenue en faisant le produit des probabilités d'occurrence des événements-sommets de ces arbres de défaillance.

Conclusion

Bien que la seconde approche de modélisation ait été conçue à l'origine pour pallier un problème calculatoire, elle a l'avantage d'inciter le fiabiliste à étudier de près les systèmes-supports et donc à identifier plus aisément les défaillances de cause commune pouvant toucher plusieurs systèmes de sûreté. *In fine*, les arbres d'événements construits à partir de cette méthode permettent d'avoir une meilleur connaissance du fonctionnement global de l'installation.

3.4 SIMULATION STOCHASTIQUE

Une des premières techniques utilisées pour l'évaluation qualitative et quantitative des arbres de défaillance a été la simulation de Monte Carlo. La simulation offre l'avantage de pouvoir estimer les mesures de certaines grandeurs de la sûreté de fonctionnement avec relativement peu de moyens informatiques. Elle permet surtout de réaliser ces estimations sur des systèmes de taille et/ou de complexité trop importantes pour pouvoir être traités par des méthodes analytiques.

L'évaluation du fonctionnement d'un système est déduite de la simulation du comportement des composants qui le constituent.
L'idée est de «mimer» la réalité. Supposons que nous souhaitions connaître une estimation de l'indisponibilité d'un système à un instant t donné. Nous allons simuler (nous verrons par la suite comment) un modèle fiabiliste représentatif du système, c'est-à-dire que nous allons faire subir aux composants fictifs du modèle des défaillances et des réparations de manière aléatoire, mais conforme aux lois régissant réellement ces événements. A l'aide de notre modèle, nous savons à tout instant dans quel état est le système (en fonctionnement ou en panne). Nous simulons ainsi N fois notre modèle dans les mêmes conditions initiales et noter n le nombre de fois où il était en panne à l'instant t. Le rapport n/N représente une estimation de l'indisponibilité du système à l'instant t.
Lorsque N tend vers l'infini, la valeur de ce rapport tend vers l'indisponibilité du système. Nous aurions pu aussi par la même occasion et au même coût (temps de simulation) évaluer la défiabilité du système sur l'intervalle [0, t].
Les autres grandeurs ou mesures de la sûreté de fonctionnement comme le MTTF, MTBF, la maintenabilité, ... peuvent également être évaluées de la même manière.

La façon la plus simple de réaliser une simulation est d'utiliser un échéancier. L'échéancier est une liste de couples (temps ; action) ordonnée en fonction des instants de réalisation des diverses actions à prendre en compte. La première action que nous intégrons à l'échéancier est l'action d'arrêt de simulation pour un temps correspondant au temps de mission du système : ($T_{Mission}$; *StopSimul*) .

Supposons que le système soit modélisé à partir d'un arbre de défaillance, chaque événement de base e_i représente un composant c_i pouvant tomber en panne en fonction d'un taux de défaillance donné et être réparé en fonction d'un taux de réparation. A l'instant t=0, tous les composants fonctionnent (aucun des événements de base n'est réalisé). Nous tirons au hasard (Cf. § suivant), pour chaque composant c_i le délai D_{def} au bout duquel il tombe en panne et nous ajoutons à l'échéancier (D_{def}+t, *Panne*(c_i)).
La simulation d'une histoire est ensuite réalisée par la boucle suivante :

Tant que l'échéancier n'est pas vide :

- Récupérer la première action de l'échéancier : (T_{act}, *action*)
- Supprimer cette action de l'échéancier
- Fixer le temps courant t à T_{act} : $t = T_{act}$
- Suivant l'action à réaliser :
  - *action* = *Panne*(c_i) : mettre en panne le composant c_i, tirer au hasard le délai D_{rep} au bout duquel il sera réparé et ajouter ce temps à l'échéancier (D_{rep}+t, *Reparation*(c_i)).
  - *action* = *Reparation*(c_i) : réparer le composant c_i, tirer au hasard le délai D_{def} au bout duquel le composant sera de nouveau en panne et ajouter ce temps à l'échéancier (D_{def}+t, *Panne*(c_i)).
  - *action* = *StopSimul* : à l'aide du modèle du système et de la connaissance de l'état des composants élémentaires (fonctionnement ou panne), déterminer si le système fonctionne ou non. Si le système ne fonctionne pas, incrémenter de 1 la variable n qui correspond au nombre d'histoires ayant abouti à la panne du système. Arrêter la simulation au terme de la mission.

Nous disposons pour calculer les différents délais de manière aléatoire, d'une part d'un générateur de nombres pseudo-aléatoires qui nous fournit un nombre uniformément distribué entre 0 et 1 et, d'autre part d'un taux de défaillance ou de réparation du composant. Dans le cas de la loi exponentielle de paramètre λ, le délai est généré par inversion de la fonction de répartition définie ci-dessous :

$$F(t) = 1 - e^{-\lambda t} \qquad [3\text{-}4]$$

F(t) renvoie un nombre compris entre 0 et 1. Posons u = F(t), où u est le nombre équiréparti entre 0 et 1. L'équation précédente peut s'écrire :

$$t = -\frac{\ln(u)}{l} \quad \text{, qui est le délai cherché} \qquad [3\text{-}5]$$

Nous pouvons, par l'inversion de la fonction de répartition, prendre en compte de nombreuses lois de probabilité (Weibull, Erlang, Lognormale, ...). De la même manière, nous pourrions aussi considérer que le temps de réparation des composants sont des délais fixes.

La simulation nous permet d'enrichir les modèles de base afin de prendre en considération :

- certaines interdépendances entre les composants de base comme la politique de maintenance, des délais de récupération, ...
- les portes des arbres de défaillance n'ayant pas d'équivalent booléen : porte délai, séquentielles, etc

Un certain nombres d'outils utilisent le formalisme des diagrammes de fiabilité à des fins de simulation stochastique, soit en fixant directement les dépendances entre les composants de base (politique de maintenance et de réparation, défaillance de cause commune, redondance passive ou reconfiguration, gestion des stocks et plus généralement des ressources, ...) [Opt90], soit en traduisant les diagrammes de fiabilité en réseaux de Petri stochastiques [SCH02].

Le reproche habituellement fait à cette méthode est qu'elle ne fournit pas de résultats exacts, mais seulement des estimations. C'est là doublement un faux problème. Premièrement, ces estimateurs ne sont pas nécessairement ponctuels mais sont souvent exprimés sous forme d'intervalles de confiance. Deuxièmement, que peut signifier un résultat donné avec plusieurs chiffres significatifs alors que les données de base, qui ont servi à le calculer, sont entachées d'incertitudes souvent importantes ? *A contrario*, les techniques de simulation de Monte-Carlo permettent de prendre en compte ces incertitudes en les propageant à travers le modèle jusqu'à l'événement final qu'on cherche à évaluer.
En fait, le vrai problème, non occulté par les personnes adeptes de ces techniques de simulation, est celui de l'estimation de l'occurrence des événements rares. Cette estimation peut requérir un nombre très important d'itérations, qui a pour conséquence une durée de simulation pouvant s'avérer prohibitive.

3.5 BIBLIOGRAPHIE

[AM93] J.D. Andrews and T.R. Moss. *Reliability and Risk Assessment*. John Wiley & Sons, 1993. ISBN 0-582-09615-4.

[ARB89] E. Arbaretier. SOFIA : génération et traitement automatique des réseaux de fiabilité à partir de la connaissance issue des analyses fonctionnelle et dysfonctionnelle d'un système. *Actes du 2e Colloque annuel du Club Fiabex*, Nanterre, Editions EC2 , p. 9 – 21, 1989.

[DCSB95] Y. Dutuit. E. Châtelet, J. Dos Santos, T. Bouhoufani. Les diagrammes-blocs fonctionnels : une aide à la construction manuelle des arbres de défaillance – Systèmes sans boucle de réaction. *Revue Européenne Diagnostic et sûreté de fonctionnement*, Vol. 5, N°2, p. 181-200, 1995.

[DRS96] Y. Dutuit, A. Rauzy and J.-P. Signoret. *Reseda: a Reliability Network Analyser*, in Proceedings of European Safety and Reliability Association Conference, ESREL'96, pp 1947-1952, vol. 3, 1996

[DSC00] J. Dugan, K. Sullivan and D. Coppit. *Developing a Low-Cost, High-Quality Software Tool for Dynamic Fault Tree Analysis.* To appear IEEE Transaction on Reliability, March 2000.

[Gac97] L. Gacôgnes. *Eléments de Logique Flou*. Edition Hermes, ISBN 2-86601-618-1, 1997

[GH01] J. Gauthier, T. Hutinet. Atelier Cécilia - Outil de Conception et d'Analyse Système (O.C.A.S.) Manuel utilisateur. *Dassault - IXI* Version 2.5, 2002.

[HCW01] D. Huang, T. Chen and M.J. Wang. *A fuzzy set approach for event tree analysis*. Fuzzy Sets and Systens 118 pp. 153-165, 2001

[Ken91] R. Kenarangui. *Event-Tree Analysis by Fuzzy Probability*. IEEE Transactions on Reliability, Vol. 40, NO. 1, pp 120-124, 1991

[Kho96] K. Khodabandehloo. Analyses of robot systems using fault and event trees : case studies. Reliability Engineering and System Safety, vol. 53 pp 247-264, 1996

[Lib96] J. Libmann. *Eléments de sûreté nucléaire*. Les éditions de la physique, ISBN : 2-86883-274-1, 1996

[MCFB94] J.-C. Madre, O. Coudert, H. Fraisse and M. Bouissou. *Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis*. In Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94, pp 118-123, 1994.

[Opt90] Optagon, *A Monte Carlo simulation package for the availability and reliability assessment of complex systems, with specialised features for the oil and gas industry*. In web site www.bg-optagon.com. 1990-2001

[PG80] A. Pagès and M. Gondran. *Fiabilité des systèmes*, volume 39 of *Collection de la Direction des études et Recherches d'électricité de France*. Eyrolles, 1980. ISSN 0399-4198.

[Ras92] D. M. Rasmuson. *A comparison of the small and large event tree approaches used in PRAs*. Reliability Engineering and System Safety, vol. 37 pp 79-90, 1992

[RSS75] Reactor Safety Study - *An assessment of accident risks in US commercial nuclear power plants*. WASH 1400 (NUREG 74/014) US NRC, October 1975 (appelé aussi "Rapport Rasmussen")

[SCH02] J.P. Signoret, J.L. Chabot et T. Hutinet, *Hiding stochastic Petri nets behind a reliability block diagrams*. To appear in prooceding of lm13/ESREL 2002, 19-21 Mars 2002.

[Ter84] K. Terpstra. *Phased Mission Analysis of maintained systems. A study in Reliability and Risk Analysis*. Thesis Netherlands Energy Research Foundation ECN, 1984.

[Vil88] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, 1988.

4. ARALIA WORKSHOP

Aralia WorkShop est un atelier de sûreté de fonctionnement axé sur les modèles booléens d'analyse des risques. Parallèlement aux actions de recherche présentées dans cette thèse, j'ai spécifié et développé cet atelier. Il a permis au logiciel Aralia de sortir du domaine de la recherche pour être connu et reconnu dans le monde industriel. Hormis les qualités intrinsèques de ce cœur de calcul, il avait besoin pour asseoir sa renommée d'une interface de saisie conviviale propre aux modèles qu'il permet de traiter.
Ce travail d'ingénierie ne s'est pas fait sans difficulté. Proposer une interface utilisateur pour Aralia demande une connaissance approfondie des BDD en général, des algorithmes et du fonctionnement d'Aralia en particulier. Ce chapitre a comme objectif de présenter succinctement les modules d'Aralia WorkShop (section 4.1), et d'une manière plus approfondie le gestionnaire de calcul (section 4.2). Une petite section sur l'organisation du logiciel (section 4.3) permettra de se rendre compte de l'importance du projet. Pour finir, j'ai grandement participé à l'étude d'une loi "Test périodique" et à la mise en œuvre des défaillances de cause commune (DCC), deux sujets développés dans les sections 4.4 et 4.5 de ce chapitre.
A l'heure actuelle, le produit Aralia WorkShop est commercialisé sous une forme limitée. En effet, seul le module permettant de saisir les arbres de défaillance (SimTree) est actuellement disponible. Les modules associés aux arbres d'événements et aux diagrammes de fiabilité sont en phase d'industrialisation et devraient être disponibles pour le congrès λμ13 qui se tiendra à Lyon en mars 2002.

4.1 MODULES DE SAISIE DES MODELES BOOLEENS

L'atelier Aralia WorkShop permet la saisie des trois principaux modèles booléens que sont les diagrammes de fiabilité, les arbres de défaillance et les arbres d'événements. Une étude de marché est en cours afin de déterminer si la réalisation d'un module permettant de saisir des réseaux de fiabilité pourrait être envisagée.
La figure suivante donne un aperçu des différents modules de l'atelier Aralia WorkShop. Le formalisme graphique des différents modèles est fidèlement respecté.

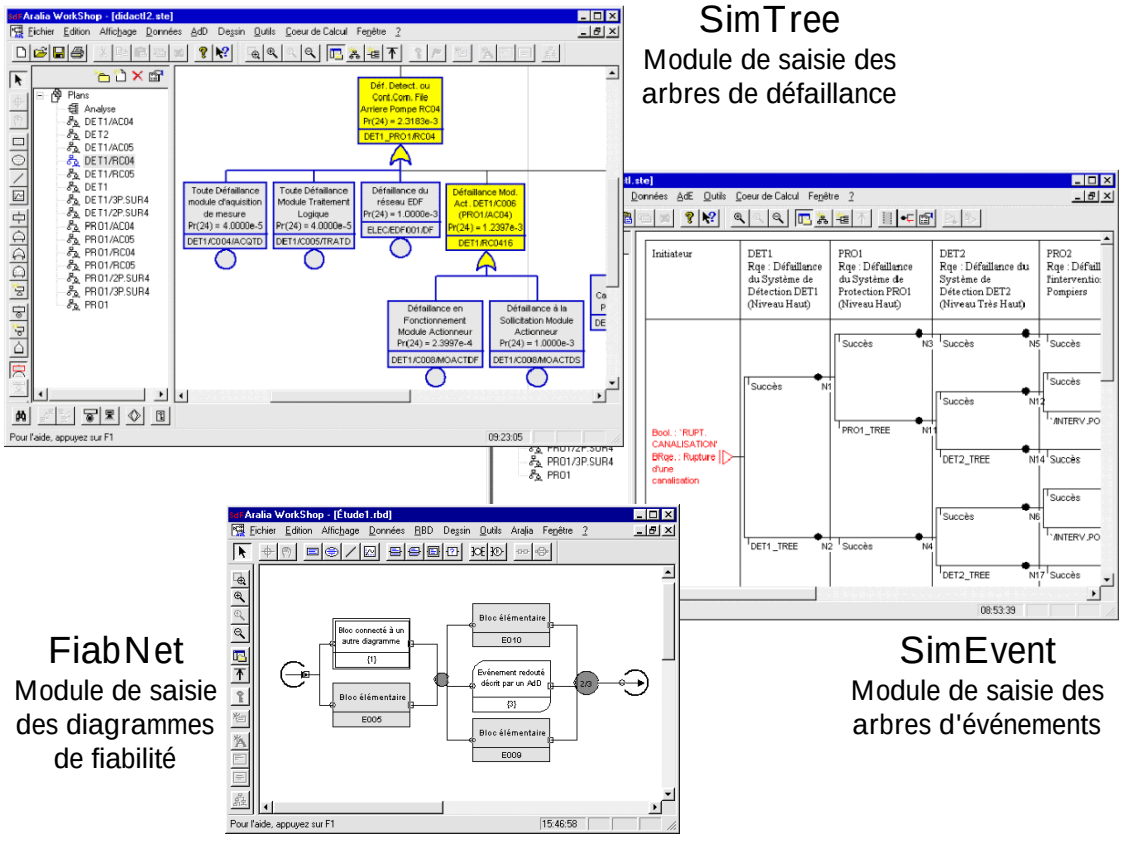


Figure 4.1 : Copie d'écran des différents modules d'Aralia WorkShop

Aralia WorkShop s'intègre de manière naturelle à Windows. Les résultats et les données peuvent être mis en forme directement au sein de Aralia WorkShop afin de générer des rapports d'analyse ou exporter vers d'autres applications comme des tableurs afin d'enrichir leurs présentations. Le presse-papier Windows est aussi opérationnel tant pour l'affichage au sein d'Aralia WorkShop d'informations provenant d'autres applications, que pour l'insertion des modèles graphiques au sein de rapports réalisés en dehors d'Aralia WorkShop.

4.1.1 LES EVENEMENTS DE BASE

Bien que les diagrammes de fiabilité utilisent la dénomination de bloc-composant comme bloc élémentaire ne demandant pas de décomposition supplémentaire, la notion d'événement de base a été conservée dans Aralia WorkShop.

Les événements de base sont donc les éléments (causes dans le cas des arbres de défaillance, composants dans le cas des blocs-diagrammes de fiabilité) qui sont considérés comme terminaux d'un point de vue du traitement booléen. Sont associés à chaque événement de base un nom (court de préférence) , un commentaire (qui l'explicite) et une loi de probabilité permettant un traitement quantitatif de la formule booléenne générée. Les lois de probabilité disponibles au sein de Aralia WorkShop sont explicitées dans l'annexe 2.

Il est également possible de spécifier des caractéristiques pour chaque événement de base à l'aide d'attributs. Un attribut est un couple formé d'un nom et d'une valeur qui peut être une chaîne de caractères, un identificateur ou un nombre.

Dans la fenêtre de dialogue présentée Figure 4.2, les deux événements de base sélectionnés ont une loi de probabilité exponentielle de paramètre Lbd. Ils ont aussi en commun un certain nombre de caractéristiques. Ils sont tous les deux associés à des composants de type "pompe" issus du même fabricant (Vaxx). Ces deux pompes supportent difficilement des températures supérieures à 300°C.

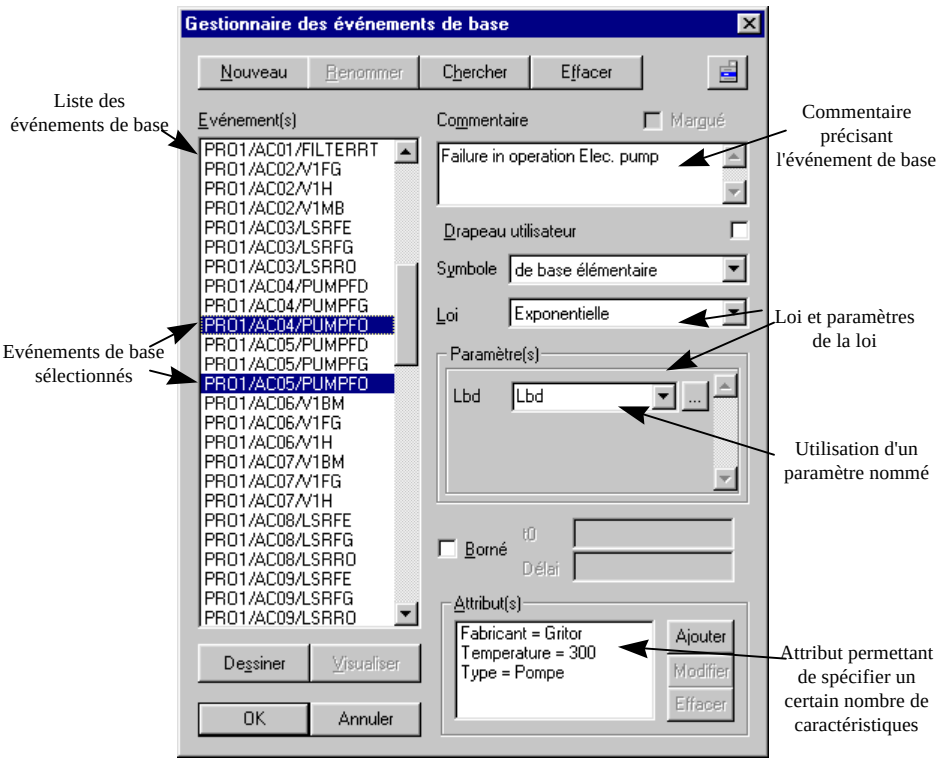


Figure 4.2 : Evénements de base dans Aralia WorkShop

Les paramètres nommés permettent d'associer le même paramètre aux lois d'un groupe de composants.

Supposons, par exemple, que l'on veuille associer une loi exponentielle de même paramètre λ à un groupe de composants. On crée un paramètre nommé de nom "Lbd", de type Taux de défaillance et de valeur λ. Puis pour l'ensemble des composants du groupe, on associe une loi exponentielle liée au paramètre nommé "Lbd" (la Figure 4.2 présente cette opération). La modification de la valeur de ce paramètre nommé est répercutée à l'ensemble des événements de base l'utilisant, ce qui évite les erreurs que l'on pourrait faire si on faisait cette opération au cas par cas.

4.1.2 SIMTREE : LES ARBRES DE DEFAILLANCE

Le module SimTree permet la saisie des arbres de défaillance. Les portes usuelles (porte Et, Ou, K-sur-N) sont disponibles afin de saisir la logique de défaillance de l'événement redouté considéré. Il convient de relier les portes les unes aux autres à l'aide d'un lien. Les autres connecteurs booléens d'Aralia (mis à part les quantificateurs existentiel et universel) sont également disponibles au sein de ce module.
Les renvois (ou transferts) identiques permettent d'alléger l'arbre en factorisant les parties identiques, mais aussi de scinder un arbre en plusieurs tronçons logiques répartis sur différentes pages, ce qui facilite la génération de rapports. Les renvois (ou transferts) semblables permettent de schématiser rapidement les installations redondantes. Ce module permet de les prendre en compte et offre une solution originale pour les développer rapidement avant de lancer les calculs.
Il est possible d'afficher en plus de l'identificateur de la porte (nom de la variable associée à la porte) et de son commentaire, d'autres informations spécifiques comme le résultat d'un dernier calcul sur cette porte, la liste des renvois associés à un identificateur, la page où se situe l'identificateur d'un renvoi, ... Toutes ces informations facilitent la lecture de l'arbre de défaillance.
Ce module traduit directement les arbres de défaillance en formules booléennes au format Aralia.

4.1.3 SIMEVENT : LES ARBRES D'EVENEMENTS

Le module SimEvent permet la saisie des arbres d'événements. La construction d'un arbre d'événements consiste à définir l'événement initiateur à prendre en compte, puis à ajouter ou supprimer des systèmes de sûreté et enfin à ajouter, supprimer ou modifier les branchements de l'arbre d'événements.
On définit les événements redoutés des branches de non-fonctionnement d'un système de sûreté à partir d'un événement de base, d'un arbre de défaillance ou d'un diagramme de fiabilité. Suivant la branche dans laquelle on se situe, il est possible d'affecter des événements redoutés différents pour un même système de sûreté. Un événement redouté par défaut est associé à chaque système de sûreté, ce qui facilite la saisie de l'arbre.
Ce module génère les arbres d'événements ainsi que les formules booléennes associées aux défaillances des systèmes de sûreté, ainsi que la cible de calcul (séquence spécifique ou ensemble de séquence) au format textuel Hévéa, afin que ce dernier puisse les transformer en formules booléennes.

4.1.4 SIMDFIAB : LES DIAGRAMMES DE FIABILITE

Le module SimDFiab permet la saisie des diagrammes de fiabilité. Il crée par défaut les nœuds source et but du diagramme de fiabilité. Il est ensuite possible d'ajouter des blocs qui peuvent être soit des blocs hiérarchiques (qui font référence à un autre diagramme de fiabilité), soit des blocs élémentaires (qui correspondent aux événements de base des arbres d'événements). Des outils permettent de les arranger rapidement dans des configurations série ou parallèle. Il est également possible de faire les liaisons entre blocs manuellement pour les architectures plus complexes. Les connecteurs K-sur-N sont aussi disponibles.
Ce module traduit directement les diagrammes de fiabilité en formules booléennes au format Aralia en utilisant l'algorithme présenté au paragraphe 3.1.3..

4.1.5 INTEROPERABILITE DES MODELES

La principale idée originale de cette interface est l'idée d'interopérabilité des modèles. En effet, quel que soit le modèle utilisé, nous manipulons une formule booléenne. La définition d'une variable non élémentaire est une formule booléenne pouvant provenir de n'importe lequel des trois formalismes. Il est possible de cette manière de mélanger les modèles.
Le formalisme généralement employé pour les arbres d'événements est le précurseur de cette idée. Les événements redoutés associés à ses branches d'échec sont décrits par des arbres de défaillance. Il est facile d'imaginer un concept équivalent pour les diagrammes de fiabilité. Le fonctionnement attendu du système est décrit de manière macroscopique à l'aide d'un (ou de plusieurs) diagrammes de fiabilité. La défaillance associée à chacun des blocs-sous-systèmes peut être représentée à l'aide d'un arbre de défaillance.
Mais l'idée d'interopérabilité des modèles va encore plus loin. Elle englobe la notion de renvoi (ou de transfert) des arbres de défaillance, ainsi que la notion de décomposition hiérarchique des blocs-diagrammes de fiabilité.
Elle permet, dans l'absolu, de considérer les trois modèles sur un même pied d'égalité, aucun des modèles n'étant prédominant vis-à-vis des autres. Dans la réalité, les choses ne sont pas aussi simples à mettre en œuvre. Si les arbres de défaillance et les diagrammes de fiabilité (et d'une manière plus générale les réseaux de fiabilité) se traduisent en formules booléennes de manière relativement aisée, les arbres d'événements posent plus de problèmes, car ils peuvent être utilisés comme des gestionnaires de formules booléennes (Cf. chapitre 5), ce qui rend leur traduction en formules booléennes plus difficile à réaliser.

4.2 GESTIONNAIRE DE CALCUL

4.2.1 LES FONCTIONS DE CALCUL DE ARALIA

La Figure 4.3 schématise les différentes structures de données manipulées par Aralia (cylindre), les algorithmes permettant de passer d'une structure de données à une autre (cartouche) et les calculs disponibles pour une structure de données (rectangle arrondi). Les flèches d'épaisseur moindre correspondent aux algorithmes pouvant engendrer des approximations.

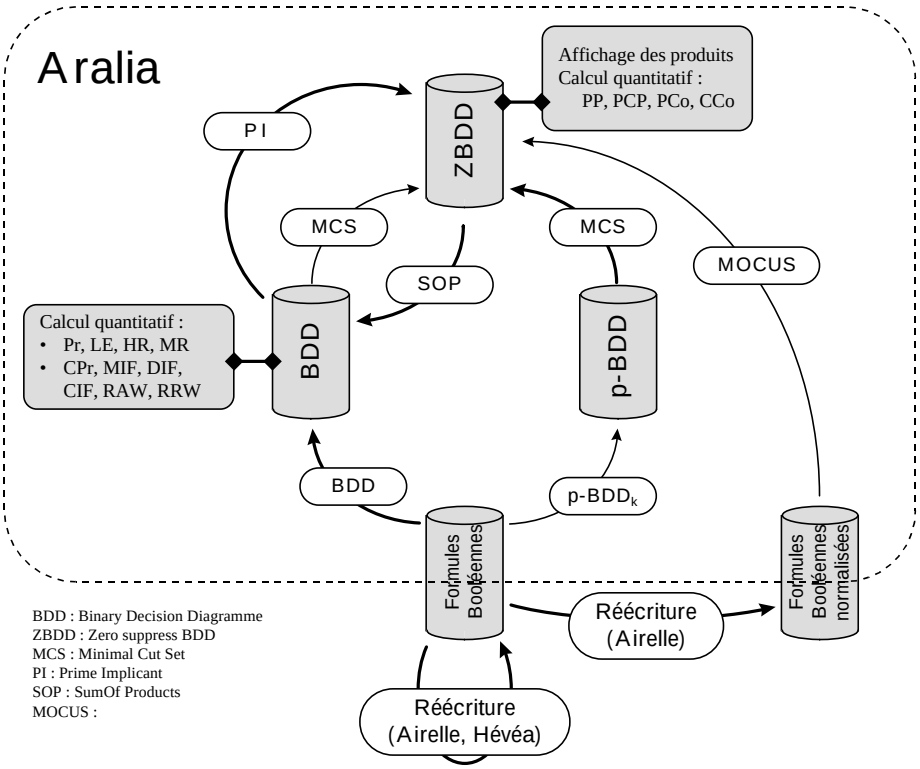


Figure 4.3 : L'architecture de calcul d'Aralia

La meilleur manière d'appréhender ce schéma est de décrire les algorithmes :

- L'algorithme BDD calcule, on s'en serait douté, le BDD d'une formule (Cf. chapitre 2.6).
- L'algorithme PI calcule les implicants premiers de la formule codée par le BDD considéré. Une variante de cet algorithme notée PI_k, renvoie uniquement les implicants premiers d'ordre inférieur ou égal à k.
- L'algorithme MCS calcule les coupes minimales de la formule codée par le BDD considéré. Une variante de cet algorithme notée MCS_k, renvoie uniquement les coupes minimales d'ordre inférieur ou égal à k.
- L'algorithme MOCUS a été présenté dans le chapitre 2.5. Il a besoin pour fonctionner d'une formule booléenne normalisée (construite à partir de littéraux positifs ou négatifs, de conjonctions et de disjonctions), ce qui est réalisable à l'aide d'un script de réécriture d'Airelle (Airelle est un module de réécriture fortement couplé à Aralia, offrant des fonctionnalités d'un langage de type shell). Il permet le calcul des coupes minimales prépondérantes d'une formule booléenne. Son efficacité dépend des approximations que l'on est prêt à faire.
- L'algorithme p-BDD a été proposé par Rauzy dans [DR98, Rau00b]. Il s'appuie sur un cadre algébrique visant à calculer, via les BDD, une approximation de la fonction F étudiée. Intuitivement, ces approximations sont obtenues en considérant les fonctions F^+_k qui sont telles que les coupes minimales de F^+_k sont celles de F à l'ordre k. Elle a été utilisée avec succès pour l'étude du système d'arrêt d'urgence d'un réacteur nucléaire [CDLR98]. Elle ne s'applique toutefois que lorsque les coupes prépondérantes de la fonction F sont courtes (d'ordre 5 ou 6 au plus). Cet algorithme doit être couplé à MCS afin d'obtenir les coupes minimales d'ordre inférieur ou égal à k. Il n'y a aucune garantie sur le BDD intermédiaire généré par cet algorithme. En résumé, nous avons, $MCS(p\text{-}BDD_k(F)) = MCS_k(BDD(F))$
- L'algorithme SOP calcule le BDD d'une formule donnée sous la forme d'une somme de produits (Sum-Of-Products) c'est-à-dire à partir d'un ZBDD.

Les résultats qualitatifs sont obtenus directement par une lecture du ZBDD.

Dans Aralia, les calculs quantitatifs qui suivent s'appliquent uniquement sur les BDD :

- Probabilité Pr(*S*,*t*) correspond à la probabilité que le système *S* soit en panne à l'instant *t*, c'est-à-dire à l'indisponibilité du système *S*.
- Probabilité conditionnelle définie par CPr(*S*,*e*,*t*) = Pr(*S*|*e*, *t*). Il s'agit donc de la probabilité que le système *S* soit en panne sachant que le composant *e* est en panne à l'instant *t*.
- Facteurs d'importance : Les facteurs d'importance permettent d'évaluer le rôle d'un composant dans la fiabilité ou la disponibilité d'un système. La notion de composant est à prendre au sens large. Un composant peut être décrit par n'importe quelle fonction booléenne. Les calculs effectués par Aralia ne sont donc pas limités aux seuls événements terminaux, ou groupes d'événements terminaux. Il faut préciser qu'il existe pas moins de cinq algorithmes au sein d'Aralia pour calculer chaque facteur d'importance. Ces algorithmes se différencient par leur efficacité, leur robustesse et le type de composants qu'ils peuvent prendre en compte. Le lecteur désireux d'étudier la définition, la justification ou le développement mathématique de ces différents facteurs d'importance peut se reporter à [DR00a].
    - Facteur d'importance Marginal (MIF)
    - Facteur d'importance de Diagnostic (DIF)
    - Facteur d'importance Critique (CIF)
    - Facteur d'Augmentation de Risque (RAW)
    - Facteur de Diminution de Risque (RRW)
- Approximation de la fiabilité : La disponibilité d'un système *S* au temps *t* est la probabilité qu'il soit en fonctionnement au temps *t*. La fiabilité de *S* au temps *t* est la probabilité que *S* ait marché continûment de 0 à *t* (inclus). Pour les systèmes non-réparables, ces deux notions se confondent. Dans le cas contraire, on ne peut qu'évaluer la disponibilité. Les deux premières grandeurs fiabilistes ci-après ne s'appliquent que si les lois associées aux variables terminales sont des lois exponentielles ou GLM (Cf. Annexe 2). Ces notions sont présentés dans le détail dans [DRS99a].
    - Lambda Equivalent (EL)
    - Fiabilité de Murchland (MR)

Il convient de préciser que la dénomination "de Poincaré" utilisée par la suite est un abus de langage, puisque le développement de Sylvester-Poincaré donne des résultats exacts, s'il est mené jusqu'au bout. Compte tenu de la nature hautement combinatoire de ce dernier, on se limite généralement au premier terme du développement, en étant conscient des approximations ainsi induites.
Des calculs quantitatifs peuvent être obtenus directement à partir des ZBDD.

- Probabilité de Poincaré PP(*S*,*t*) correspond à la somme des probabilités des coupes minimales de *S* à l'instant *t*.
- Probabilité conditionnelle de Poincaré PCP(*S*,*e*,*t*) correspond à la somme des probabilités des coupes minimales de *S* contenant *e* (*e* étant un événement terminal) à l'instant *t*.
- Contribution de Poincaré PCo(*S*,*e*,*t*) pondère la probabilité conditionnelle de Poincaré PCP(*S*,*e*,*t*) par la probabilité de Poincaré PP(*S*,*t*) du système

Pour tous ces calculs, on peut associer aux paramètres des lois de probabilité des valeurs variant aléatoirement suivant des distributions fixées (actuellement disponible dans Aralia : lognormale et uniforme). Le principe des études de traitement des incertitudes et de sensibilité est d'effectuer une simulation de Monte-Carlo sur le calcul de la probabilité de l'événement sommet. Chaque histoire de cette simulation comprend trois étapes :

1. on tire pseudo-aléatoirement la valeur des paramètres des lois en fonction de leur distribution
2. pour chaque instant considéré, on calcule les probabilités des événements terminaux,
3. on calcule enfin la valeur de l'événement sommet.

On recommence la même opération N fois, afin d'obtenir un échantillon statistiquement représentatif.

4.2.2 LES FONCTIONS DE CALCUL DANS ARALIA WORKSHOP

Le schéma Figure 4.3 montre qu'il y trois manières relativement différentes d'appréhender une formule booléenne avec Aralia :

- La première consiste à calculer le BDD codant la formule booléenne puis, si on souhaite des informations qualitatives, à calculer un ZBDD codant soit les implicants premiers (à l'aide de PI), soit les coupes minimales (à l'aide de MCS).

- La seconde consiste à calculer les coupes minimales d'une formule à l'aide de l'algorithme MOCUS. Il convient alors de prétraiter la formule afin de la rendre compatible avec cet algorithme. A partir du ZBDD, si l'on désire faire des calculs applicables uniquement sur les BDD, il convient de le calculer à l'aide de SOP.
- Enfin, la dernière consiste à calculer les coupes minimales tronquées à l'ordre k à l'aide d'un $pBDD_k$ suivi d'un MCS. On transforme le ZBBD en BDD à l'aide de SOP, si l'on désire faire des calculs nécessitant un BDD.

Ces trois approches sont accessibles directement au sein de Aralia WorkShop. Lorsque l'on souhaite faire un calcul, on a la possibilité de préciser l'approche suivant laquelle le calcul va être effectué. Les options de calcul, l'ordonnancement des appels des différents algorithmes de Aralia, la récupération des résultats sont autant de points qui diffèrent suivant l'approche considérée.

De plus, Aralia WorkShop propose un certain nombre de fonctionnalités qui ne sont pas accessibles directement depuis Aralia. Il s'agit principalement des traitements qualitatifs et quantitatifs des défaillances de cause commune qui sont détaillés dans la section 4.5 et des calculs de sensibilité.
Ces derniers consistent à modifier des paramètres de calcul, tels que la valeur d'un paramètre nommé, la probabilité d'un événement ou d'un ensemble d'événements de base, de manière contrôlée (pas de manière aléatoire comme pour les études de sensibilité avec propagation d'incertitudes), puis à calculer une grandeur fiabiliste (probabilité ou approximation de la fiabilité) sur la variable sommet à un instant t. Le but de ces études est de pouvoir juger de la sensibilité d'un composant (ou d'un groupe de composant) au sein d'une installation.

Tout ce qui précède ne se réfère qu'au paramétrage possible permettant de faire des calculs sur une seule variable sommet.
Le gestionnaire de calcul d'Aralia WorkShop permet de définir ce qu'on appelle des cas d'analyse. Un cas d'analyse contient toutes les informations nécessaires au lancement d'un calcul. A savoir, la cible du calcul (Sommet d'un arbre de défaillance, sélecteur de séquences d'un arbre d'événements, diagramme de fiabilité) ainsi que l'approche choisie et les paramètres de calcul à prendre en compte.

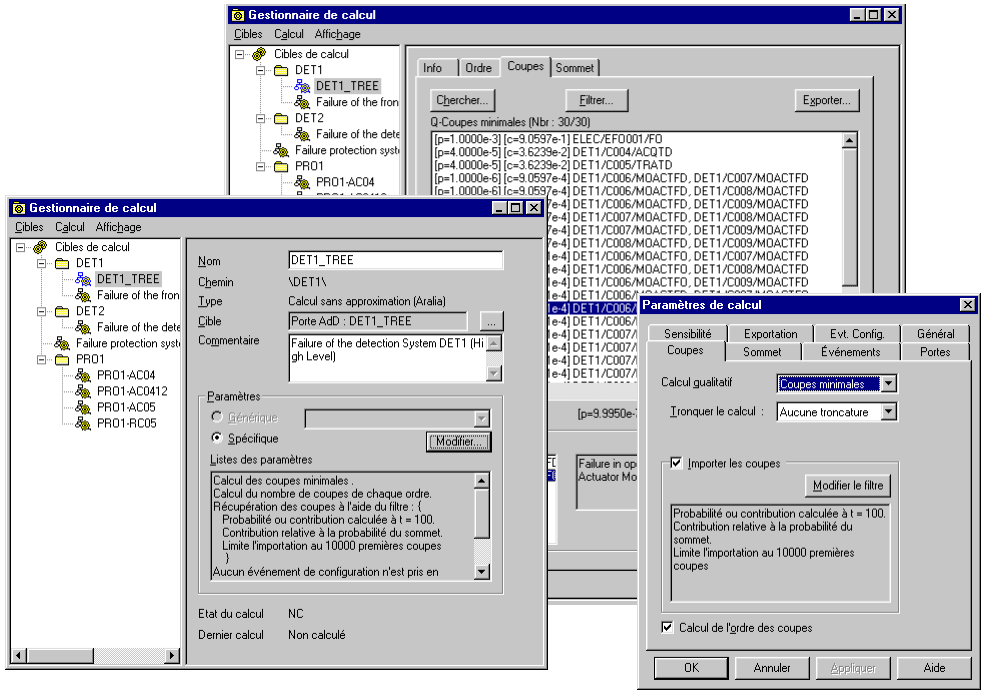


Figure 4.4 : Gestionnaire de calcul de Aralia WorkShop

Ces cas d'analyse sont accessibles à partir d'une arborescence semblable à celle du gestionnaire de fichier. Les menus offrent des commandes qui permettent de les modifier, de les supprimer ou de faire des sélections par critères de recherche. Il est possible de modifier les paramètres de calcul et/ou la cible du calcul. Une fois défini le cas d'analyse, il suffit alors de le sélectionner afin de lancer les calculs associé. Les résultats obtenus sont mémorisés au sein de banque de résultats.

4.3 ORGANISATION LOGICIELLE

Comme le montre la Figure 4.5, Aralia WorkShop est un projet de taille industrielle.
Sans tenir compte des cœurs de calcul (Aralia, Réséda, Hévéa, Airelle, ...) Aralia WorkShop est composé d'une vingtaine de modules, ce qui représente plus de 200 000 lignes de code réparties dans un peu plus de 1000 fichiers.

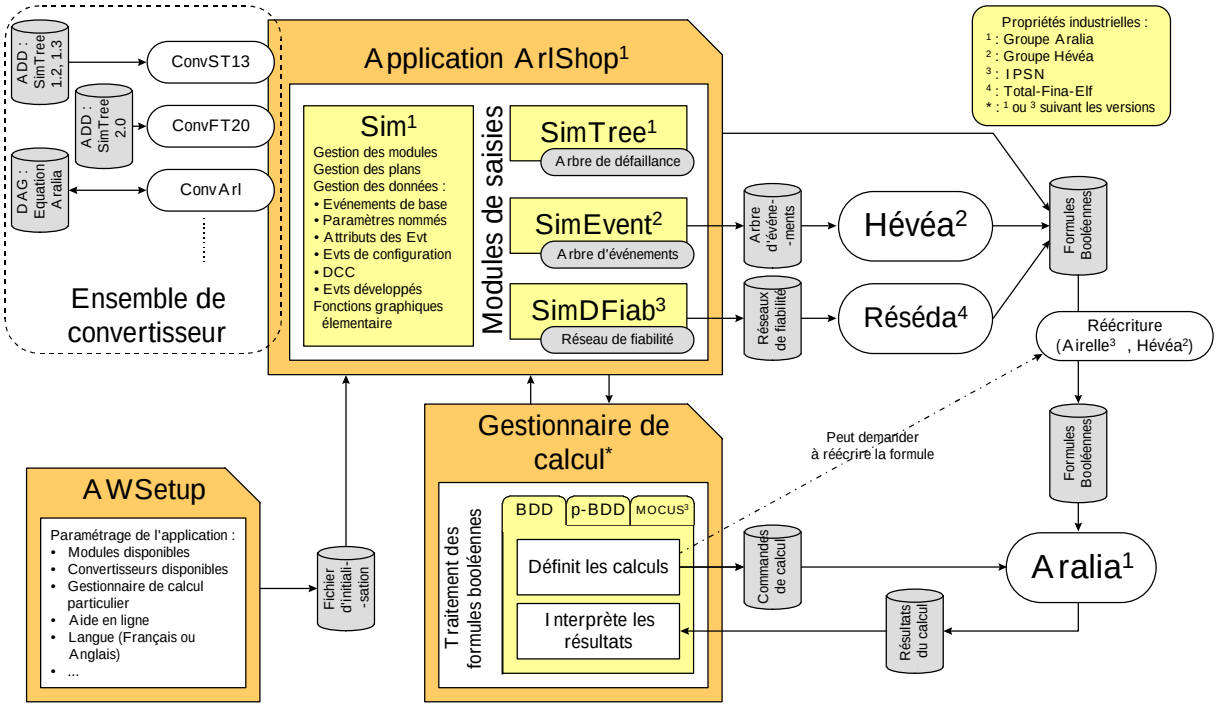


Figure 4.5 : Organisation logicielle

La particularité d'Aralia WorkShop est que ses modules n'appartiennent pas tous aux mêmes propriétaires. Si la structure de base du logiciel appartient au groupe Aralia, certains membres de ce groupe ont supporté seuls les développements de modules leur tenant à cœur. C'est le cas de l'IPSN qui a financé successivement le module "diagramme de fiabilité", le gestionnaire de calcul dans sa version la plus évoluée, ainsi qu'Airelle et l'algorithme MOCUS que l'on trouve dans Aralia.
Cette diversité de propriétés industrielles a conduit à favoriser la modularité dans Aralia WorkShop de manière significative. Les modules peuvent s'acquérir indépendamment les uns des autres. Il suffit de préciser au démarrage d'Aralia WorkShop quels sont les modules à charger.
Le module AWSetup se charge de paramétrer Aralia WorkShop. Plus précisément, il permet de spécifier :

- les modules de saisie (SimTree, SimEvent, SimDFiab et même, depuis peu, des modules spécifiques à telle ou telle société)
- les convertisseurs disponibles (ces convertisseurs permettent d'ouvrir les fichiers des anciennes versions, mais aussi les descriptions des modèles au format textuel Aralia et/ou Hévéa),
- le gestionnaire de calcul à considérer (il existe sous différentes versions plus ou moins conviviales),
- la langue (Anglais ou Français)
- ...

4.4 LOI "TESTS PERIODIQUES"

Il existe dans Aralia deux lois de tests périodiques. La première représente plus un composant soumis à une politique de remplacement périodique, qu'un composant dont on teste le fonctionnement puis qu'on répare s'il est défaillant. De plus, la première loi considère le remplacement comme instantané et sûr. L'expression de la disponibilité ou de la fiabilité d'un tel composant est donnée ci-après :

$$A(t) = \begin{cases} 1 - \exp^{-l.t} & \text{si } t \le t \\ 1 - \exp^{-l.((t-t) \bmod q)} & \text{si } t > t \end{cases} \qquad [4\text{-}1]$$

avec l : le taux de défaillance du composant,
t : la date du premier test
q : la période de remplacement (intervalle de temps entre 2 remplacements consécutifs)

Le domaine de validité limité de cette loi nous a incités à définir dans [DRST99] une loi de test périodique plus réaliste et à l'intégrer au sein d'Aralia. Cette seconde loi est plus complète. Elle prend en compte un nombre important de paramètres regroupés et définis dans le tableau suivant :

Par.	Définition	Limites
λ	Taux de défaillance en cours de fonctionnement	Taux ≥ 0
λ^*	Taux de défaillance durant le test	Taux ≥ 0
μ	taux de réparation (une fois la panne détectée)	Taux ≥ 0
θ	Premier intervalle entre tests	Temps > 0
τ	Intervalle entre deux tests consécutifs	Temps > 0
γ	Probabilité de défaillance due au déclenchement du test	0 ≤ Probabilité ≤ 1
π	Durée du test	Temps > 0
X	Indicateur de disponibilité du composant durant le test (= 0, le composant est indisponible; = 1, il est disponible)	0 ou 1
σ	Taux de couverture du test (probabilité que la panne du composant soit détectée lors du test)	0 ≤ Probabilité ≤ 1
ω	Probabilité d'oubli de reconfiguration (après le test et après la réparation)	0 ≤ Probabilité ≤ 1

Tableau 4-1 : Les différents paramètres de la loi "Test périodique"

Les deux schémas ci-après montrent comment interviennent ces différents paramètres.

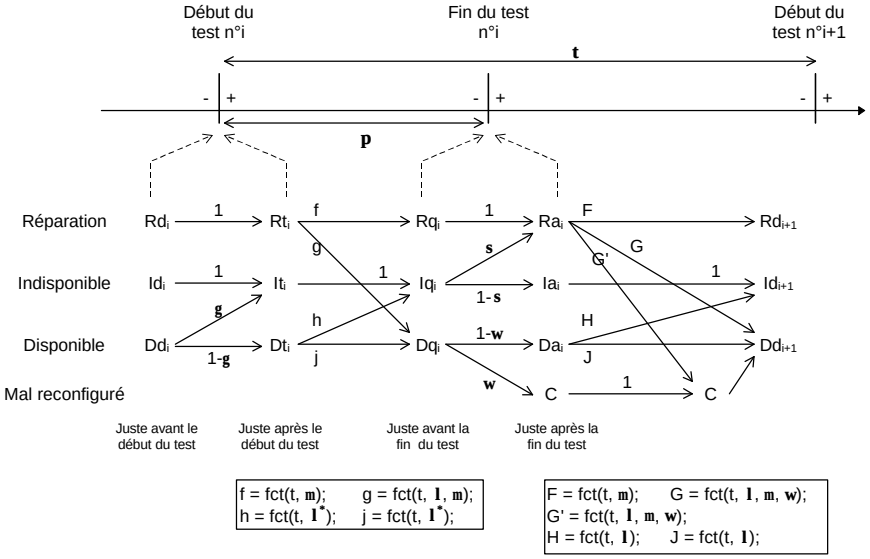


Figure 4.6 : Visualisation du rôle des différents paramètres de la loi "Test périodique" (1/2)

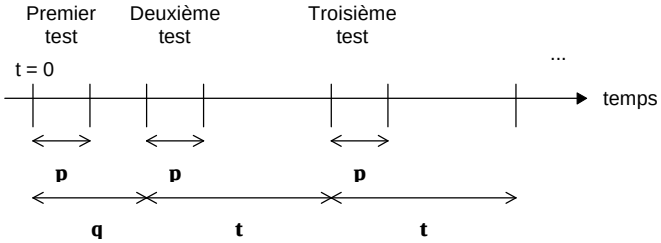


Figure 4.7 : Visualisation du rôle des différents paramètres de la loi "Test périodique" (2/2)

4.5 DEFAILLANCES DE CAUSE COMMUNE

Les défaillances de cause commune (DCC) constituent un sous-ensemble des défaillances dépendantes.
Plusieurs définitions en ont été données [AM93, Vil88]. Mais, dans la cadre de Aralia WorkShop, nous avons adopté la définition suivante : "*Défaillances de même mode, simultanées ou quasi simultanées de deux ou plusieurs composants identiques technologiquement, dans un même système et dues à une même cause*".
On considère que les défaillances de cause commune qui affectent des composants élémentaires résultent soit d'agressions externes, soit d'erreurs humaines commises à la conception, à la fabrication, au montage ou en exploitation. Elles peuvent être classées en cinq types :

- Les agressions de l'environnement (poussière, saleté, humidité, température, vibrations, conditions météorologiques, séisme, inondation, projectile, incendie, explosion, chute d'avion...).
- Les erreurs de conception.
- Les erreurs de fabrication.
- Les erreurs de montage.
- Les erreurs d'exploitation et de maintenance (Oublis de reconfiguration systématique sur un ensemble de composants identiques, ...).

Pour traiter les défaillances de cause commune, on dispose de trois méthodes au sein d'Aralia WorkShop.

1. La principe de la première méthode consiste à expliciter une défaillance de cause commune directement au sein du modèle sous la forme d'un événement de base. Les événements étant alors de nouveau considérés comme indépendants, les techniques de modélisation usuelles peuvent être employées.
2. La méthode d'analyse prévisionnelle [Vil88] cherche à mettre en évidence la potentialité de défaillance de cause commune. C'est une méthode qualitative se basant sur une exploitation des coupes minimales visant à étudier leur sensibilité à des risques de défaillances de cause commune. Le principe relativement simple est développé de la manière suivante :
   - On définit des attributs représentant les causes communes potentielles des défaillances (localisation du composant, fabricant, sensibilité à l'inondation, à l'incendie, au séisme, ...)
   - Pour chaque composant du système, on donne une valeur à chacun des attributs (par exemple, localisation = H7, Inondation = Sensible, Incendie = ">120°C", ...)
   - Une fois les coupes minimales obtenues, on recense celles qui sont sensibles à une catégorie ou à un attribut particulier.

   Il est ainsi possible de mettre en évidence des coupes potentielles d'ordre 1 dues à une défaillance de cause commune.
3. La dernière solution est d'utiliser une méthode paramétrique permettant de prendre en compte de manière empirique ces défaillances de cause commune. Ces méthodes sont issues du domaine nucléaire. Une étude [DD98] a été menée au sein du groupe Hévéa par l'IPSN pour recenser les principales méthodes paramétriques employées dans le nucléaire. Le principe de base de ces méthodes est expliqué dans la suite de ce paragraphe.

Parmi les types de méthodes présentées dans [DD98], Aralia WorkShop permet de prendre en compte, *a priori*, toutes les méthodes paramétriques sans choc. Dans cette catégorie, on retrouve les méthodes du facteur β, des Lettres Grecques Multiples, du facteurs α, ... Le fondement de tous ces modèles paramétriques est qu'ils s'appliquent aux groupes constitués de k composants identiques.
Soit un groupe de DCC composé des défaillances A, B, C et D qui peuvent correspondre à des défaillances de pompes identiques (même fabricant, même numéro de série). L'idée de base développée dans ces approches est que la méthode habituelle de construction du modèle n'est pas modifiée, mais qu'en revanche, lors de la phase de traitement, des équations booléennes supplémentaires sont générées afin de prendre en compte ces DCC. On

transforme chaque défaillance incluse dans un groupe de DCC par la conjonction des défaillances de cause commune où l'événement est présent.
Exemple :

A := ($A\bar{B}\bar{C}\bar{D}$ | $AB\bar{C}\bar{D}$ | $A\bar{B}C\bar{D}$ | $A\bar{B}\bar{C}D$ | $ABC\bar{D}$ | $AB\bar{C}D$ | $A\bar{B}CD$ | ABCD);
$A\bar{B}\bar{C}\bar{D}$ signifie la défaillance de A mais ni celle de B, ni celle de C ni celle de D.

De la même façon, on écrit :

B := ($\bar{A}B\bar{C}\bar{D}$ |$AB\bar{C}\bar{D}$ |$\bar{A}BC\bar{D}$ |$\bar{A}B\bar{C}D$ |$\bar{A}BCD$ |$AB\bar{C}D$ |$\bar{A}BCD$ |ABCD);
C := ($\bar{A}\bar{B}C\bar{D}$ |$A\bar{B}C\bar{D}$ |$\bar{A}BC\bar{D}$ |$\bar{A}\bar{B}CD$ |$\bar{A}BCD$ |$A\bar{B}CD$ |$\bar{A}BCD$ |ABCD);
D := ($\bar{A}\bar{B}\bar{C}D$ |$A\bar{B}\bar{C}D$ |$\bar{A}B\bar{C}D$ |$\bar{A}\bar{B}CD$ |$\bar{A}BCD$ |$A\bar{B}CD$ |$\bar{A}BCD$ |ABCD);

Cette transformation n'est pas du ressort d'Aralia. C'est l'interface Aralia WorkShop qui s'en occupe. Les membres du groupe Aralia n'ayant pas jugé opportun d'ajouter au sein d'Aralia des mécanismes qu'ils n'approuvaient pas dans leur totalité. Il existe en revanche une loi non documentée dans Aralia qui permet de définir la probabilité à associer à une feuille ABCD en fonction de la loi de probabilité du groupe de DCC (ou de chaque événement du groupe) et de paramètres de répartition.

Fondamentalement, et quels que soient les composants A, B, C et D, on peut écrire :

$$\begin{aligned} 1 \quad &= p(\bar{A}\bar{B}\bar{C}\bar{D}) \\ &+ p(A\bar{B}\bar{C}\bar{D}) + p(\bar{A}B\bar{C}\bar{D}) + p(\bar{A}\bar{B}C\bar{D}) + p(\bar{A}\bar{B}\bar{C}D) \\ &+ p(AB\bar{C}\bar{D}) + p(A\bar{B}C\bar{D}) + p(A\bar{B}\bar{C}D) + p(\bar{A}BC\bar{D}) + p(\bar{A}B\bar{C}D) + p(\bar{A}\bar{B}CD) \\ &+ p(ABC\bar{D}) + p(AB\bar{C}D) + p(A\bar{B}CD) + p(\bar{A}BCD) \\ &+ p(ABCD) \end{aligned} \qquad [4\text{-}2]$$

$p(A\bar{B}\bar{C}\bar{D})$ est la probabilité que A seul soit défaillant parmi A, B, C, D, c'est-à-dire qu'un composant soit défaillant sur les quatre (noté : p(1#4)). Il en est de même pour $p(\bar{A}B\bar{C}\bar{D})$.
En fait, on peut écrire :

$$1 = p(0\#4) + 4\times p(1\#4) + 6\times p(2\#4) + 4\times p(3\#4) + p(4\#4) \qquad [4\text{-}3]$$

D'une manière plus générale, pour un groupe de k composants, on a :

$$1 = \sum_{i=0}^{k} \left(C_k^i \times p(i\#k) \right) \qquad [4\text{-}4]$$

Quelle que soit la méthode de quantification de DCC utilisée, le principe reste le même : définir les p(i#k) en fonction d'une loi de probabilité associée à un événement de base du groupe.
A un groupe de k composants, on définit k paramètres de répartition notés $\rho_1,\rho_2,...,\rho_k$ tels que :

$$p(i\#k) = r_i \times Q_t \qquad [4\text{-}5]$$

Q_t étant l'indisponibilité intrinsèque de chaque composant

La loi "facteur" d'Aralia prend en paramètre un facteur (r_i : valeur réelle) et une loi de probabilité ($Q(t)$). Sa valeur au temps t est égale à la valeur de la loi de probabilité au temps t multipliée par r_i. Elle permet de prendre en considération les défaillances de cause commune à partir du moment où l'interface se charge de la transformation des événements soumis à une défaillance de cause commune et du calcul des différents paramètres de répartition utilisés.

Les différents modèles paramétriques sans choc se différencient dans la définition mathématique des paramètres de répartition. Notons ici les deux principales méthodes utilisées :

1. Méthode du facteur β (β factor model)
   Cette méthode a été introduite par Fleming [FMK83]. C'est probablement le modèle le plus répandu pour traiter les DCC. La principale raison de son succès est son extrême simplicité d'utilisation. Le principe est le suivant : soit un seul composant tombe en panne du fait d'une défaillance indépendante, soit tous les composants du groupe de DCC tombent en panne simultanément, avec une seule et même cause de défaillance.
   Le paramètre β est défini comme étant égal au pourcentage de défaillances résultant d'une cause commune. Les paramètres de répartition pour Aralia sont alors définis par :

$$\begin{aligned} &p(1\#k) = (1-b)\times Q_t \\ &p(j\#k) = 0 \qquad \text{pour } 2 \le j < k \\ &p(k\#k) = b \times Q_t \end{aligned} \qquad [4\text{-}6]$$

2. Méthode des Lettres Grecques Multiples (Multiple Greek Letter model)
   Cette méthode est une généralisation du modèle du facteur β lorsque l'on considère un nombre de composants supérieur à deux.
   Les paramètres de répartition pour Aralia sont alors définis par :

$$p(i\#k) = \frac{1}{C_{k-1}^{i-1}} \times \left(\prod_{j=1}^{i-1} b_j \right) \times (1 - b_i) \times Q_t \qquad \text{avec } b_0 = 1 \text{ et } b_k = 0 \qquad [4\text{-}7]$$

4.6 ALOES FAIT-IL PARTIE D'ARALIA WORKSHOP ?

Dans cette présentation de l'atelier Aralia WorkShop, nous n'avons à aucun moment parlé de Aloès. Nous verrons dans la partie III consacrée à ce projet qu'Aloès est couplé à Aralia pour les traitements des arbres de défaillance et à Réséda pour celui des réseaux de fiabilité. Il serait assez naturel de coupler Aloès à Aralia WorkShop.
Aloès n'est actuellement qu'un outil expérimental. C'est un cœur de calcul fonctionnant à l'aide d'un interpréteur de commande. Le langage permettant de décrire un modèle d'allocation est assez verbeux. Pour pouvoir être utilisé par des ingénieurs fiabilistes, il reste à valider l'outil dans un contexte industriel, puis à spécifier et à développer une interface pour Aloès.
Cette interface faciliterait l'utilisation d'Aloès à deux niveaux :

- La construction du modèle.
- Le lancement et le suivi du calcul.

Pour le lancement et le suivi des calculs, une approche *calculette* est à privilégier. L'utilisateur doit pouvoir charger un modèle, faire varier les paramètres de ce dernier, tester une solution, lancer un calcul d'optimisation, l'arrêter quand bon lui semble, visualiser la solution en cours, revenir en arrière, ... Ces fonctionnalités imposent une connexion forte avec Aloès. Il n'est pas nécessaire qu'il soit couplé avec Aralia WorkShop. Un fonctionnement autonome permettrait de l'utiliser avec ces propres outils.
En revanche, l'aide à la construction d'un modèle peut être un module d'Aralia WorkShop a plus d'un titre. Un arbre de défaillance ou un réseau de fiabilité construit à l'aide de Aralia WorkShop pourrait être utilisé comme base d'un modèle Aloès. Les informations spécifiques aux composants du système (comme son coût d'achat, son poids, ...) pourraient être spécifiées à l'aide des attributs des événements de base. Des cas d'analyse spécifiques à Aloès permettraient de définir les variables de décision du modèle, ainsi que les équations reliant toutes ces informations. Ces cas d'analyse généreraient alors un modèle au format Aloès et ce modèle serait traité directement à l'aide de la calculette Aloès.

4.7 BIBLIOGRAPHIE

[AM93] J.D. Andrews and T.R. Moss. *Reliability and Risk Assessment*. John Wiley & Sons, 1993. ISBN 0-582-09615-4.

[CDLR98] S. Combacon, Y. Dutuit, A. Laviron, and A. Rauzy. *Comparison between two tools (Aralia and ESCAF) applied to the Study of the Emergency Shutdown System of a Nuclear Reactor*. Proceedings of the International Conference of Probabilistic Safety Assessment and Management, PSAM'4, volume 2, pages 1019-1024, New-York, 1998. Springer Verlag.

[DD98] F. Ducamp, C. Dutuit *Représentation et quantification des DCC - Cahier des charges pour Hévéa* Note d'étude EPS1/REP900/NOTE/98-012 1998

[DR00a] Y. Dutuit and A. Rauzy. *Efficient Algorithms to Assess Components and Gates Importances in Fault Tree Analysis*. Reliability Engineering and System Safety, 72(2):213–222, 2000.

[DRS99a] Y. Dutuit, A. Rauzy, and J.-P. Signoret. *Evaluation of systems reliability by means of binary decision diagram.* In Proceedings of the Probabilistic Safety Assessment Conference, PSA'99, volume 1, pages 521–528. American Nuclear Society, 1999. ISBN 0-89448-640-3.

[DRST99] Y. Dutuit, A. Rauzy, J.P. Signoret et P. Thomas. *Disponibilité d'un système en attente et périodiquement testé.* Actes du 3ème Congrès International Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita 99, 1999, pp 367-375.

[FMK83] K. Fleming, A. Mosheh and A. Kelley. *On the analysis of Dependant Failures in Risk Assessment and Reliability Evaluation*. Nuclear Safety, val 24, n°5, Sept-Oct 1983.

[Vil88] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, 1988.

Partie II :
Projet Hévéa

Depuis une vingtaine d'années et la parution du rapport WASH-1400 [RSS75], l'utilisation conjointe de la technique des arbres d'événements et de celle des arbres de défaillance (ET/FT) s'est progressivement imposée dans l'ingénierie nucléaire. Un arbre d'événements code un ensemble de séquences d'événements. Chaque séquence commence par un événement initiateur, passe par un certain nombre d'événements génériques et aboutit dans un état caractéristique du système modélisé (panne récupérée, panne non récupérée, …). Les événements génériques sont souvent définis par des arbres de défaillance. La quantification des arbres d'événements pose deux problèmes mathématiques difficiles, liés à l'interdépendance des événements génériques. Il peut exister deux types de dépendances entre ces événements :

- Des dépendances fonctionnelles : les arbres de défaillance définissant deux événements génériques peuvent partager des événements terminaux. Si tel est le cas, le calcul de la probabilité d'occurrence d'une séquence les contenant tous les deux doit tenir compte de cette dépendance.
- Des dépendances temporelles : une séquence dans l'arbre d'événements décrit une trajectoire du système modélisé. L'ordre dans lequel les événements surviennent peut jouer un rôle important. Par exemple, l'instant de démarrage d'un composant en redondance passive dépend de l'instant de la détection de la panne du composant principal. Il s'ensuit que la probabilité d'occurrence d'une séquence ne peut pas toujours être calculée en faisant le produit des probabilités d'occurrence des événements la composant.

Nous avons proposé une traduction des arbres d'événements en équations booléennes (suivant en cela de nombreux auteurs, par exemple [Pap98]). Cette traduction permet de résoudre le premier des deux problèmes ci-dessus. Dans ce cadre, la prise en compte des dépendances temporelles entre événements génériques se ramène au problème général de la prise en compte de phénomènes temporels dans les arbres de défaillance.
Ce travail s'est effectué dans le cadre d'un partenariat entre différentes structures (le LaBRI d'une part, l'IPSN, Technicatome, Elf-EP et IXI d'autre part). Ce partenariat s'était fixé comme objectif la réalisation d'un logiciel de traitement d'arbres d'événements associé à Aralia/SimTree.

Le logiciel comprend en fait deux parties : d'une part, le cœur de calcul Hévéa qui compile les arbres d'événements en équations booléennes destinées à être traitées par Aralia ; d'autre part, une interface graphique conviviale permettant la saisie des arbres d'événements et la connexion avec le cœur de calcul.
Le projet Hévéa a comporté trois phases distinctes, présentées respectivement dans les chapitres 5, 6 et 7 :

1. La première a eu comme objectif la formalisation mathématique des arbres d'événements et leur traduction en formules booléennes [TR99], ainsi que la réalisation du logiciel Hévéa. Les études réalisées par les partenaires industriels du projet ont montré les difficultés inhérentes au traitement des formules booléennes produites (en raison de la taille de ces dernières).
2. La deuxième phase a consisté à étudier les moyens à mettre en œuvre pour traiter ces problèmes d'explosion combinatoire [DMRST00], [DPRT00]. Je me suis consacré à la mise au point de différentes méthodes de réécriture des formules booléennes permettant de simplifier leur traitement. Antoine Rauzy s'est, dans le même temps, chargé de la réalisation d'un algorithme à la "MOCUS" (Cf. chapitre 2.5) au sein de Aralia, ainsi que de la mise au point de stratégies spécifiques d'utilisation des diagrammes binaires de décision. A l'issue de ces travaux, tous les arbres d'événements fournis par les partenaires industriels ont pu être quantifiés de manière approchée et de manière exacte.
3. La dernière phase du projet a été consacrée à l'étude des différentes approximations utilisées dans les logiciels classiques de quantification des arbres d'événements booléens. Ces approximations ont été simulées à l'aide des algorithmes disponibles au sein de Aralia. Deux études probabilistes de sûreté de taille conséquente ont servi de référence pour la comparaison entre les différentes approximations et les résultats exacts.

5. TRAITEMENT DES ARBRES D'EVENEMENTS

Le but du projet Hévéa était de réaliser un logiciel, support des EPS, utilisant une approche mixte ET/FT (arbres d'événements couplés à des arbres de défaillance).
Dans ce contexte, les événements redoutés associés aux branches de non-fonctionnement sont décrits par des arbres de défaillance.

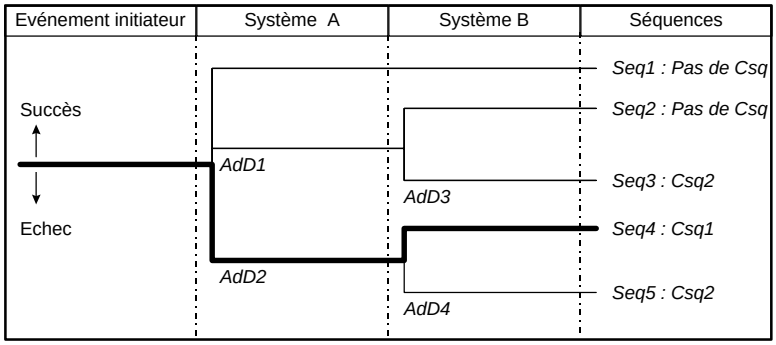


Figure 5.1 : Exemple d'arbre d'événements

La première section de ce chapitre présente le cadre général du formalisme permettant la traduction des arbres d'événements en formules booléennes. Les événements de configuration peuvent être utilisés dans les arbres d'événements afin de modifier la structure des arbres de défaillance d'une séquence particulière. L'intérêt et la prise en compte de ces événements de configuration sont discutés dans la section 5.2. Hévéa permet de décrire les séquences que l'on souhaite traduire en formules booléennes à l'aide d'un langage inspiré des expressions régulières. Ce langage est décrit dans la section 5.3. Pour finir, la section 5.4 est consacrée à la présentation des tests réalisés. Il s'agit de deux Etudes Probabilistes de Sûreté (EPS) qui ont permis d'étudier :

1. Les techniques de réécriture permettant de traiter les formules booléennes générées par Hévéa.
2. La validité des approximations probabilistes généralement utilisées par les logiciels classiques de traitement des EPS.

5.1 TRADUCTION EN FORMULES BOOLEENNES

Nous proposons de considérer un arbre d'événements comme un graphe orienté acyclique dont les sommets représentent des états du système modélisé et dont les arêtes sont étiquetées par des formules booléennes. Les variables de ces formules représentent, comme dans les arbres de défaillance, la survenue d'événements élémentaires. Le système passe d'un état s dans un état t en empruntant l'arête étiquetée par la formule f si cette dernière est réalisée.
Le graphe associé à l'arbre de la Figure 5.1 est représenté sur la Figure 5.2 (les arêtes étant orientées de gauche à droite).

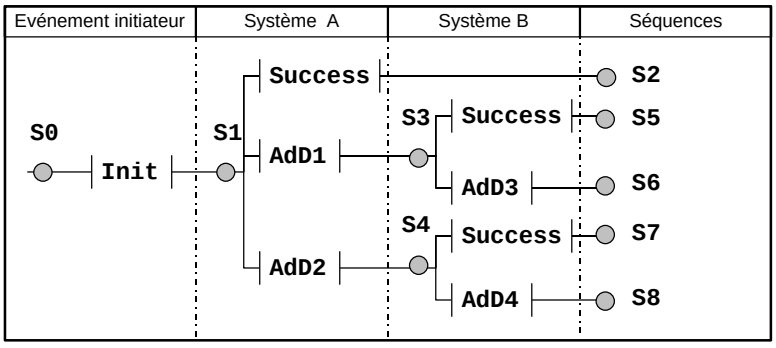


Figure 5.2 : Représentation du formalisme au sein des arbres d'événements

Sur la Figure 5.2, les branches supérieures sont étiquetées par le mot clé `success`. C'est une convention d'écriture : ces branches sont en fait étiquetées par la conjonction des négations des formules étiquetant les autres

branches sortant du sommet origine de l'arête. Par exemple, la branche supérieure sortant de l'état S1 doit être étiquetée par la formule ($\neg AdD_1 \wedge \neg AdD_2$).
L'interprétation d'une séquence $S_0[f_1]S_1 \ldots S_{n-1}[f_n]S_n$, dans le graphe associé à un arbre d'événements, est simplement la conjonction des formules étiquetant les arêtes de ce chemin, c'est-à-dire ($f_1 \wedge \ldots \wedge f_n$). L'interprétation d'un ensemble de chemins est la disjonction des interprétations des chemins composant cet ensemble.
Dans la méthode des arbres d'événements, les systèmes de sûreté sont représentés au sein de colonnes. Le formalisme état-transition que propose Hévéa offre une alternative agréable à cette représentation. Ainsi, le diagramme de la Figure 5.3 contient toutes les informations nécessaires pour traduire les séquences de l'arbre d'événements représentées en formules booléennes.

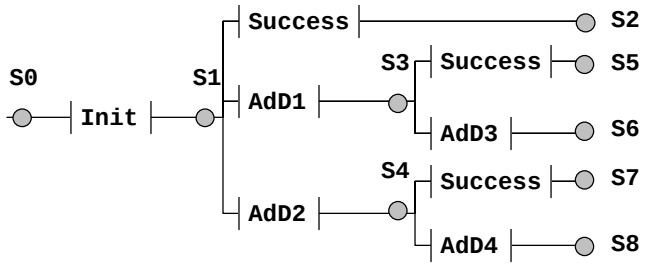


Figure 5.3 : Formalisme Hévéa indépendant des arbres d'événements

Le formalisme Hévéa a un pouvoir d'expression plus important que celui des arbres d'événements. Il permet des constructions telles que celle de la Figure 5.4. Les règles de traduction restent les mêmes que celles présentées au début de la section, puisqu'elles ne définissent que la traduction d'une séquence en formule booléenne.

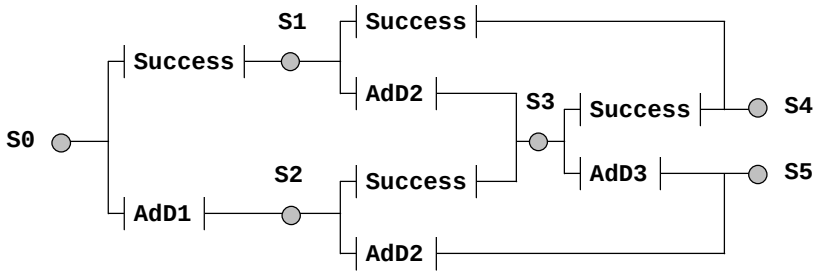


Figure 5.4 : Diagramme orienté acyclique au formalisme Hévéa

La Figure 5.4 représente un système qui à partir de l'état initial S0, peut aboutir à l'un des deux états terminaux S4 et S5 représentant respectivement un état de bon fonctionnement et un état de panne. On atteint l'état S3 en cas de succès (S0→S1) d'un sous-système SSyst1 dont la panne est décrite à l'aide de AdD1 suivi de l'échec (S1→S3) d'un sous-système SSyst2 ou, inversement, en cas d'échec du SSyst1 suivi du succès de SSyst2. Il s'agit donc d'un système composé de deux sous-systèmes SSyst1 et SSyst2 en série pouvant être éventuellement secourus par un troisième sous-système SSyst3, en cas de panne.

L'exemple précédent peut être représenté à l'aide du formalisme des arbres d'événements en utilisant la notion de renvois de séquences qui sont le pendant des renvois identiques des arbres de défaillance.

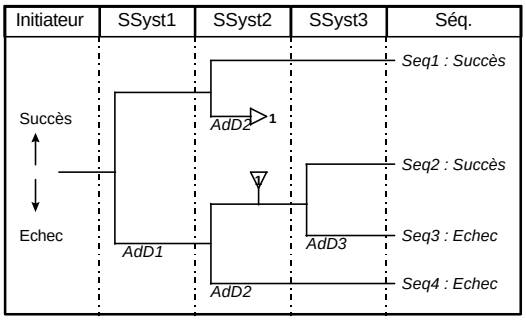


Figure 5.5 : Utilisation de renvois dans les arbres d'événements

L'utilisation d'un diagramme orienté acyclique comme base du formalisme Hévéa permet en tout cas de traiter convenablement les ensembles de séquences aboutissant à une conséquence donnée. Cela permet d'utiliser cette

dernière en tant qu'événement-initiateur d'un autre arbre d'événements. La Figure 5.6 explicite cette idée utilisée dans certain logiciels traitant des EPS. Certaines séquences des deux arbres d'événements A et B aboutissent à l'événement initiateur de l'arbre d'événements C.

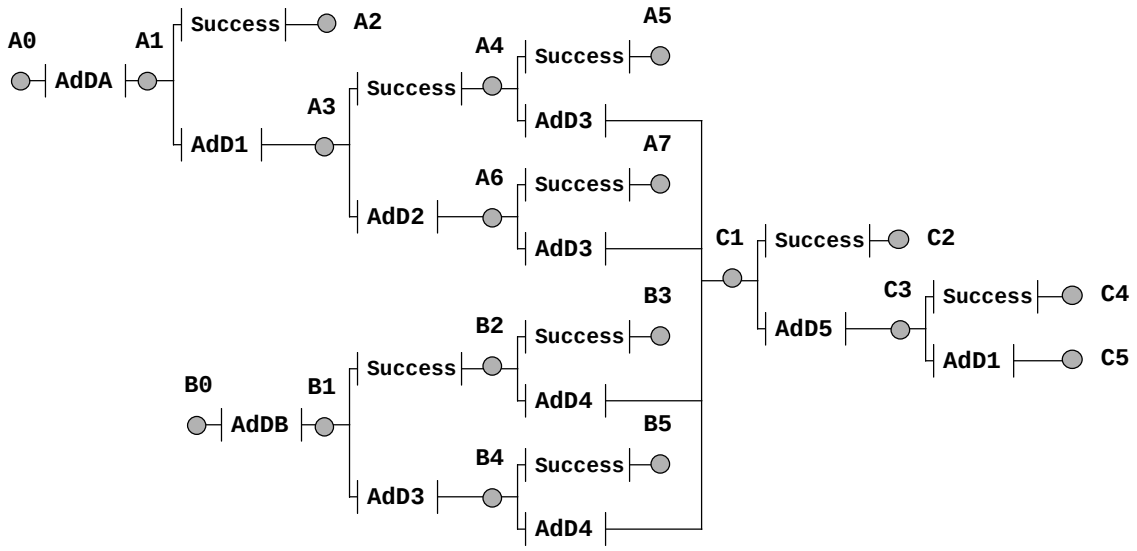


Figure 5.6 : Cascade d'arbres d'événements

Dans ce cas, les logiciels historiques scindent le calcul en deux phases. Dans un premier temps, ils calculent une probabilité d'occurrence associée à la conséquence en additionnant les probabilités des séquences qui y aboutissent. Dans un second temps, ils associent cette probabilité à l'événement initiateur du second arbre d'événements. Cette analyse en deux phases simplifie considérablement les calculs, car elle n'étudie pas le problème dans sa globalité. Elle aboutit à ne pas prendre en considération les dépendances fonctionnelles pouvant exister entre les systèmes utilisés pour empêcher l'apparition de cette conséquence et les systèmes mis en œuvre pour en réduire l'importance. Ce n'est bien entendu pas le cas du formalisme du projet Hévéa qui permet de traiter également ce problème de dépendance fonctionnelle.

5.2 LES EVENEMENTS DE CONFIGURATION AU SEIN DES AE

Les événements de configuration («house events») sont des constantes booléennes. Ils permettent de paramétrer les séquences, et ainsi de factoriser les descriptions. Leurs valeurs sont toujours connues au moment du calcul.

5.2.1 UTILITE

Lorsque les descriptions de deux événements redoutés sont proches, les événements de configuration permettent de les factoriser en un seul arbre en précisant leur spécificité.

Par exemple, considérons un problème d'inondation dans un local pourvu de 4 pompes. Suivant la taille de la fuite d'eau, 2 ou 3 pompes parmi les 4 doivent fonctionner pour extraire l'eau du local. On peut décrire les deux événements redoutés ("Inondation du local (panne de plus d'une pompe)" et "Inondation du local (panne de plus de deux pompes)") à l'aide d'un seul arbre de défaillance.
Lorsque Cfg1 est positionné à Vrai (Grosses fuites), G1 devient une porte 2 parmi 4. Lorsque deux pompes sont en panne, l'événement redouté survient. Dans le cas contraire (Cfg1 positionné à Faux), G1 devient une porte 3 parmi 4. L'événement redouté survient lorsque 3 pompes sur les 4 sont en panne.

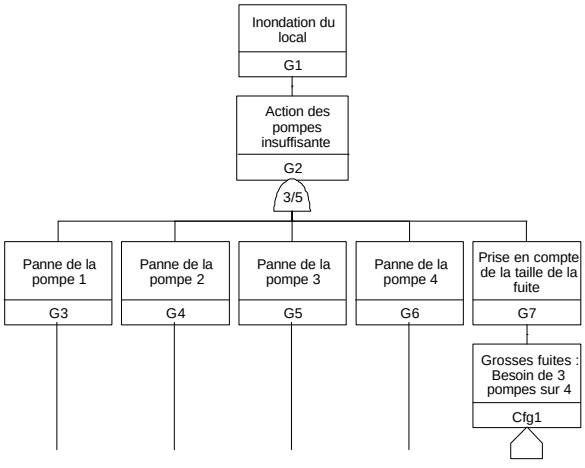


Figure 5.7 : Arbre de défaillance avec événement de configuration

Le positionnement (à vrai ou à faux) d'un événement de configuration le long des séquences peut alors se faire à deux endroits différents :

1. soit sur une arête : l'événement de configuration agit alors comme un modificateur local de la fonction booléenne associée à cette arête ;
2. soit sur un sommet : l'événement de configuration agit alors comme un modificateur des fonctions booléennes de toutes les arêtes-filles de ce sommet. Cet événement de configuration permet, lors du déroulement de la séquence accidentelle, d'avoir une idée de ce qui s'est passé en amont. On propage à droite le positionnement de l'événement de configuration.

Cette idée de propagation d'information le long des séquences est une conséquence directe de la méthode de modélisation des "arbre d'événements avec conditions limites" (LET). Cette méthode est présentée au paragraphe 3.3.4. Elle aboutit à la construction d'arbres d'événements comprenant de nombreux systèmes de sûreté. L'analyse de la défaillance d'un système de sûreté dépend de la branche en cours et donc de la séquence envisagée.
Dans la Figure 5.8, la défaillance du système de sûreté SSyst dépend du succès ou de l'échec du système support A. Cette dépendance est prise en compte dans AdD2 sous la forme de l'événement de configuration CfgA qui valide ou invalide des branches de l'arbre en fonction de la valeur de CfgA. L'événement de configuration CfgA est propagé "à vrai" lorsque le système Support A est en panne.

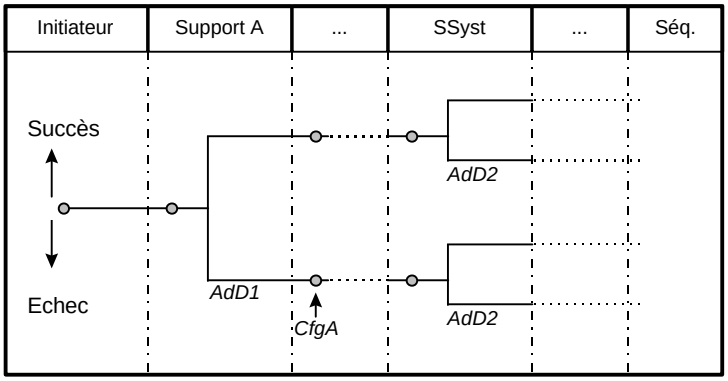


Figure 5.8 : Propagation d'événement de configuration

A chaque arête, il est donc associé un contexte, correspondant à un ensemble d'événements de configuration fixés "à vrai" ou "à faux". Le système passe d'un état *s* dans un état *t* en empruntant l'arête étiquetée par la formule *f* et le contexte *c* si *f* sachant *c* est réalisée.
La propagation des événements de configuration à droite ne s'applique qu'à des arbres. Elle s'applique plus difficilement dans le cas des diagrammes orientés acycliques que l'on peut construire à l'aide du formalisme de Hévéa. Considérons l'exemple de la Figure 5.6. Si nous propageons CfgA à partir de A6 et ¬CfgA à partir de B4, nous pouvons difficilement faire des choix sur la manière de propager ces événements de configuration opposés au moment où les deux séquences se recoupent (C1). Pour cette raison, le modèle est dit invalide si les contextes (ensemble d'événements de configuration positionnés à vrai ou à faux) provenant des différents chemins aboutissant à un état ne sont pas identiques.

5.2.2 SEMANTIQUE FORMELLE

On associe à chaque état et à chaque arête deux ensembles d'événements de configuration *V* et *F* disjoints. *V* (respectivement *F*) représente les événements de configuration positionnés à Vrai (resp. à Faux).
On note $f_{S\neg 1,T\neg 0}$ la formule *f* dans laquelle on a substitué la constante 1 à tous les événements de configuration de l'ensemble *S* et la constante 0 à tous ceux de l'ensemble *T*.

On note :

- $r(S)$, un chemin commençant par l'état *S*
- $S_i > (V_s, F_s) \bullet [f < (V_f, F_f)] \bullet \rho(S_j)$, l'arête passant de l'état s_i à l'état s_j, positionnant à vrai (respectivement à Faux) les événements de configuration de l'ensemble V_S (resp. F_S) et étiquetée par la formule $f_{Vf\leftarrow 1,Ff\leftarrow 0}$.
- ε, la séquence vide
- $CS||\sigma ; (V,F)||$, la formule associée à la séquence σ dans un contexte (V,F)
- $CS||\Sigma||$, la formule associée à un ensemble de séquences Σ.

Ces conventions étant fixées, on peut définir formellement la sémantique d'un ensemble de séquences Σ (*CS*‖Σ‖) à l'aide des équations récursives suivantes :

$$CS\left\| \{s_1,...,s_n\} \right\| = CS\left\| s_1 \; ; (\varnothing,\varnothing) \right\| \vee \ ... \ \vee \ CS\left\| s_n \; ; (\varnothing,\varnothing) \right\|$$

$$CS\left\| S_i \rangle (V_s,F_s) \bullet \left[f \langle (V_f,F_f) \right] \bullet r(S_j) \; ; (V,F) \right\|$$

$$= f_{V \cup V_S \cup V_f \setminus F_S \setminus F_f \leftarrow 1,\ F \cup F_S \cup F_f \setminus V_S \setminus V_f \leftarrow 0}$$

$$\wedge \ CS\left\| r(S_j) \; ; (V \cup V_S \setminus F_S \ , \ F \cup F_S \setminus V_S) \right\|$$

$$CS\left\| e \; ; (V,F) \right\| = 1$$

De cette manière les ensembles *V* et *F* d'un contexte sont toujours disjoints.
Les événements de configuration associés aux arêtes (locaux) sont prioritaires par rapport aux événements de configuration propagés.

$f_{S\leftarrow 1,T\leftarrow 0}$ est traduit vers les modèles booléens de la manière suivante :

$$f_{S\leftarrow 1,\ T\leftarrow 0} = f' \wedge \prod_{x \in S} (x = 1) \wedge \prod_{y \in T} (y = 0)$$

avec *f'* la copie de *f* en renommant les variables intermédiaires, mais pas les feuilles (événements de base).

5.3 EXPRESSIONS REGULIERES DECRIVANT LES ENSEMBLES DE SEQUENCES

Comme nous l'avons vu dans la première section de ce chapitre, l'intérêt des arbres d'événements est de pouvoir effectuer des calculs sur une ou plusieurs séquences accidentelles. Il est donc nécessaire de disposer d'un moyen pratique et efficace pour caractériser les séquences pertinentes.
Nous avons proposé d'utiliser à cette fin un langage de description inspiré des expressions régulières.

Les expressions régulières que nous avons considérées sont définies de la façon suivante :

- Si Σ est un ensemble d'états, alors {Σ} est une expression régulière élémentaire qui permettra de définir les chemins qui commencent ou qui aboutissent à un état de Σ.
- Si E1 et E2 sont des expressions régulières, alors E1.E2 est une expression régulière qui représente l'ensemble de toutes les séquences de la forme σ.τ où σ est une séquence de E1 se terminant par un certain état T et τ est une séquence de E2 commençant par T.
- Si E1 et E2 sont des expressions régulières, alors E1,E2 est une expression régulière qui représente l'union des séquences de E1 et de E2.
- Si E est une expression régulière, alors E∗ est une expression régulière. Cette expression représente l'ensemble de toutes les séquences composées d'aucune, d'une ou de plusieurs séquences de E.

Le point d'interrogation ? représente n'importe quel état.

Par exemple le sélecteur $\{S_0\}.?^*.\{S_6,S_8\}$ caractérise l'ensemble de séquences commençant par l'état S_0 et se terminant par l'état S_6 ou l'état S_8. Interprété sur le graphe de la Figure 5.3, ce sélecteur représente donc l'ensemble formé des deux séquences $S_0.[Init].S_1.[AdD_1].S_3.[AdD_3].S_6$ et $S_0.[Init].S_1.[AdD_2].S_4.[AdD_4].S_8$.
Le sélecteur $\{S_0\}.?^*.\{S_5\}$ interprété sur le graphe de la Figure 5.4 représente l'ensemble des 3 séquences suivantes : $S_0.[\neg AdD_1].S_1.[AdD_2].S_3.[AdD_3].S_5$; $S_0.[AdD_1].S_2.[\neg AdD_2].S_3.[AdD_3].S_5$; $S_0.[AdD_1].S_2.[AdD_2].S_5$
Par contre, le sélecteur $\{S_0\}.?^*.\{S_3\}.?^*.\{S_5\}$ne considère, dans l'exemple ci-dessus, que les deux premières des trois séquences passant par S_3.

A partir de ce langage de description, la traduction sous la forme d'automate normalisé (ou automate de Thompson) [WL93] ne pose pas de problème. L'annexe E présente au lecteur qui désire plus d'informations sur cette traduction. A l'aide de cet automate, il est relativement aisé de tester si une suite d'états (c'est-à-dire un chemin dans l'arbre d'événements) aboutit à un état acceptant de l'automate.
Pour récupérer l'ensemble des séquences de l'arbre d'événements reconnues par le sélecteur de séquences, on fait un parcours en profondeur à gauche de l'arbre d'événements. C'est cet ensemble de séquences qui est traduit en formules booléennes.

5.4 IMPLEMENTATION DE HEVEA

Les principes exposés dans le paragraphe précédent sont implémentés au sein du logiciel Hévéa.

Ce logiciel est un interpréteur de commandes manipulant trois types de données bien distincts :
1. les formules booléennes qui permettent, rappelons-le, de décrire les événements redoutés associés aux branches de panne des arbres d'événements,
2. les états représentant le caractère séquentiel des arbres d'événements,
3. les sélecteurs de séquences.

Le code qui suit correspond à la description de l'arbre d'événements de la Figure 5.2 au format d'entrée textuel :

```
S0 :=   case
            Init : S1
        esac;
S1 :=   case
            success : S2,
            AdD1 : S3,
            AdD2 : S4
        esac;
S3 :=   case
            success : S5,
            AdD3 : S6
        esac;
S4 :=   case
            success : S7,
            AdD4 : S8
        esac;
```

Les sélecteurs de séquence qui suivent correspondent aux différentes séquences de l'arbre d'événements.

```
Seq1 := select ?* . {S2};
Seq2 := select ?* . {S5};
Seq3 := select ?* . {S6};
Seq4 := select ?* . {S7};
Seq5 := select ?* . {S8};
```

Hévéa associe à chaque variable de sélection une formule booléenne équivalente à son ensemble de séquences.
En plus des commandes de saisie du modèle, d'affichage et de traduction en formules booléennes, Hévéa permet de vérifier la cohérence des arbres d'événements saisis, en particulier la propagation des événements de configuration et le non-déterminisme des branchements. De plus, Aralia, grâce à ces possibilités de connexion logicielle, est directement disponible au sein de Hévéa. Toutes les commandes de description des modèles booléens (équations booléennes, lois de probabilité des variables terminales, ...) ont la même syntaxe dans Hévéa que dans Aralia et elles sont toutes disponibles dans Hévéa.

Le module SimEvent permet de saisir les arbres d'événements dans un formalisme très proche d'Hévéa, ce qui en a facilité la connexion.

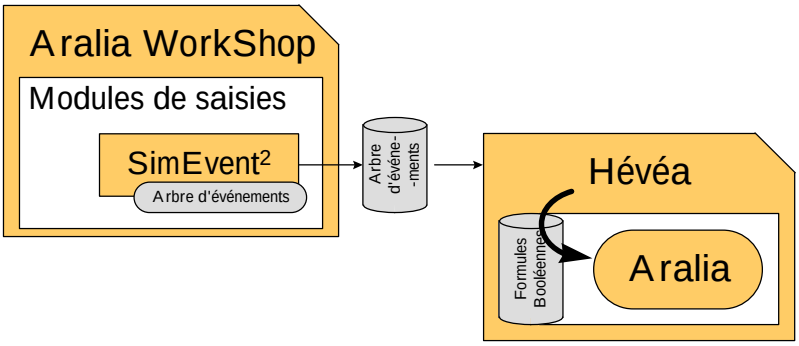


Figure 5.9 : Organisation logicielle

Hévéa est un logiciel de taille respectable totalisant environ 30 000 lignes de code.

5.5 VALIDATION DE HEVEA

5.5.1 EPS AYANT SERVI A LA VALIDATION

Hévéa (ou plus exactement SimEvent) a été utilisé dans un cadre opérationnel afin de valider les logiciels (interface et cœur de calcul), ainsi que l'approche générale (traduction vers les formules booléennes et traitement de ces formules de manière exacte).
Deux études probabilistes de sûreté ont été réalisées.

- La première fait intervenir 18 initiateurs qui aboutissent à 329 séquences. Tous les branchements (311) sont binaires (succès, échec). Les séquences sont relativement longues (faisant intervenir de nombreux systèmes de sûreté), mais les formules booléennes associées aux arbres de défaillance sont de tailles moyennes. Aucun événement de configuration n'est utilisé. 206 séquences aboutissent à la fusion du cœur. Deux types de fusion ont été envisagés. La fusion du cœur sous haute pression est le résultat de 55 séquences, tandis que 135 séquences aboutissent à une fusion du cœur en basse pression.
- La seconde, de taille plus importante, est composé de 1819 séquences provenant de 109 initiateurs. Il s'agit de la même EPS que celle étudiée dans [DPRT00, Rau02a]. Les branchements (au nombre de 1710) sont aussi tous binaires. Contrairement à la première EPS, les séquences sont relativement courtes, mais les systèmes de sûreté considérés sont de taille importante. De plus, les techniques de conditions aux limites (boundary conditions) sont utilisées pour la majorité des initiateurs (90) et de nombreux nœuds intermédiaires (448).

La saisie à l'aide de l'interface graphique et la traduction en formules booléennes par le cœur de calcul ont été validées à l'aide de ces deux EPS. En revanche, à la fin de la première phase du projet, la quantification de certaines séquences ne pouvait être réalisée par le cœur de calcul Aralia.

5.5.2 DIFFICULTE DU TRAITEMENT DES BDD

Pour chaque séquence, les résultats souhaités par les utilisateurs sont, *a minima*, la probabilité de la séquence et la liste des coupes minimales prépondérantes. Il est à noter ici que c'est effectivement les coupes minimales, et non les implicants premiers, qui sont à analyser. En effet, une séquence, pouvant comporter la réussite d'une ou de plusieurs parades, va engendrer des implicants premiers ayant des littéraux négatifs correspondant à des non pannes (donc au fonctionnement) de composants utilisés par les parades ayant fonctionné. Comme précisé au chapitre 2.2, ces littéraux négatifs ne sont pas porteurs d'informations pertinentes et alourdissent considérablement la lisibilité des résultats. Un calcul de coupes minimales semble donc plus approprié. Il est important de préciser que, dans cette analyse, les coupes calculées ne sont utilisées que d'un point de vue qualitatif. La probabilité et les éventuelles autres grandeurs quantitatives sont calculées directement à l'aide du BDD, ce qui assure l'exactitude des résultats. Pour finir, les coupes prépondérantes sont celles qui, étant très courtes (ordre inférieur ou égal à trois), doivent être analysées de manière systématique, ou celles qui contribuent le plus, au sens probabiliste, à la réalisation de l'événement considéré (contribution supérieure à 0,1 %). Cette étude devant valider une approche opérationnelle, la durée de calcul a été limitée à 10 minutes, ce qui correspond à un temps acceptable pour un utilisateur.

La première étude de validation était basée sur un calcul du BDD afin de calculer la probabilité de la séquence de manière exacte, et sur le calcul d'un ZBDD de type MCS (Cf. paragraphe 4.3) afin d'obtenir l'ensemble des coupes minimales. Cet ensemble était ensuite filtré afin de ne récupérer que les coupes prépondérantes.
Les résultats de ces expérimentations sont donnés dans les tableaux ci-dessous sous la forme du nombre de séquences traitées dans un intervalle de temps donné :

EPS1	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
BDD	321	8	0	0	0	0	0	0	329
ZBDD	230	73	10	0	0	8	8	0	329

EPS2	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
BDD	1202	149	3	14	0	0	249	202	1819

Ainsi, les BDD correspondant à 321 séquences (sur 329) de l'EPS1 ont été calculés en moins d'une seconde. Le calcul des BDD des 8 autres séquences n'a pas excédé 5 secondes. 8 calculs de coupes minimales n'ont pas abouti en moins de 10 minutes, ce qui est également le cas pour le calcul des BDD d'un quart des séquences de l'EPS2.

La colonne "Mem" correspond au nombre de séquences qui ont saturé, en moins de 10 minutes, la mémoire allouée pour le calcul. Cette mémoire est définie comme étant le nombre maximum de nœuds pouvant être créés. Cette limite a été fixée à environ 4 millions de nœuds. Cette limite n'est pas très élevée, elle aurait pu être fixée à une valeur beaucoup plus importante, mais notre contexte d'utilisation correspond à une puissance de calcul limitée en terme de mémoire. Aralia WorkShop fonctionne sur un système d'exploitation Windows et prend déjà un certain nombre de ressources. Cette limite a donc été déterminée en fonction d'un confort d'utilisation minimal sur un ordinateur disposant de 128 Mo de mémoire vive.
La machine sur laquelle ont été effectués les tests présentés dans cette partie est un ordinateur à processeur AMD Duron 700 MHz disposant de 128 Mo de mémoire vive sous Windows 98.

Si le calcul du BDD ne pose aucun problème pour la première EPS (EPS1), il n'en est pas de même pour le calcul du ZBDD de EPS1. Les résultats pour la seconde EPS (EPS2) démontrent que la méthode actuelle n'est pas satisfaisante en l'état.

5.6 CONCLUSION

Une seconde phase du projet Hévéa a été conduite afin de trouver des solutions permettant de traiter toutes les séquences de manière exacte ou approchée. Les travaux ont été scindés en deux parties :

- La première approche a consisté à tester de manière plus ou moins exhaustive toutes les solutions qui permettent de diminuer la taille du BDD. Parmi ces solutions citons principalement les heuristiques sur l'ordre des variables utilisées dans la construction du BDD et les techniques de réécriture sur les formules booléennes. Le paragraphe qui suit rend compte de ces expérimentations et conclut sur l'utilisation d'une métaheuristique qui permet le calcul du BDD de l'ensemble des séquences de ces deux EPS.
- La seconde approche a consisté à mettre en place au sein d'Aralia un algorithme "à la MOCUS" efficace. Cet algorithme utilisé dans la majorité des logiciels du commerce est présenté dans le chapitre 2.4. Son but est de calculer l'ensemble des coupes prépondérantes d'une formule booléenne. Son efficacité repose principalement sur les possibilités qu'il offre en matière d'approximations. Il permet de récupérer uniquement les coupes prépondérantes, soit en termes de probabilité, soit en termes d'ordre. Ce travail a été effectué par A. Rauzy [Rau02a].

5.7 BIBLIOGRAPHIE

[DMRST00] F. Ducamp, F. Meunier, A. Rauzy, J.P. Signoret et P. Thomas. *Traitements d'arbres d'événements : Problèmes et Solutions.* Actes du 12^{e} Colloque national de fiabilité et de maintenabilité - λμ12, pp. 269-274, 2000.

[DPRT00] F. Ducamp, S. Planchon, A. Rauzy et P. Thomas. *Handling very large event trees by means of Binary Decision Diagrams*. In Proceedings of the International Conference on Probabilistic Safety Assessment and Management, PSAM'5, vol. 3, pp. 1447-1452, 2000

[Pap98] I.A. Papazoglou, Mathematical foundations of event trees, *Reliability Engineering and System Safety*, Elsevier, vol. 61, pp. 169-183, 1998

[Rau02a] A. Rauzy. *Towards an Efficient Implementation of Mocus.* IEEE Transactions on Reliability, 2001. To appear.

[TR99] P. Thomas et A. Rauzy. *Hévéa un gestionnaire d'arbres d'événements couplé à Aralia*. Actes du 3ème Congrès internationnal Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita 99, pp. 463-473, 1999.

[WL93] P. Weis et X. Leroy. *Chapitre 15 du "Le langage CAML" : Rechercher de motif dans un texte.* InterEdition. ISBN 2-7296-0639-4. 1993.

6. REECRITURE

Nous allons présenter dans ce chapitre les différentes voies qui ont été explorées pour diminuer l'explosion combinatoire résultant des calculs de BDD ou de ZBDD pour l'ensemble des séquences des deux EPS testées.
Il convient de préciser que cette étude a été réalisée dans un contexte opérationnel. L'utilisateur a une vision presse-bouton de ce genre d'étude. La manière importe peu, seul le résultat compte. Dans ce contexte, essayer de trouver des solutions génériques est une gageure.
Il ne s'agit pas de calculer le BDD d'une formule qui pose un problème, car, dans ce cas, il est possible de "décortiquer" cette dernière afin de comprendre les raisons pour lesquelles le calcul du BDD n'aboutit pas et de trouver des solutions adaptées. La difficulté est donc bien de chercher des solutions qui permettent le calcul d'une large classe de formules booléennes.
De plus, les utilisateurs travaillent sur des ordinateurs ayant une configuration mémoire donnée qui dépend des applications utilisées. Il est raisonnable d'imposer une configuration mémoire minimale, afin d'utiliser Aralia WorkShop dans un contexte EPS. L'ordinateur-cible dispose de 128 Mo de mémoire vive ce qui est considéré comme la configuration mémoire minimale à prendre en compte. Dans ce contexte d'utilisation, les gains, en termes de mémoire, ne sont guère porteurs d'information, les utilisateurs souhaitant avant tout récupérer leurs résultats rapidement afin de pouvoir les analyser.
La (ou les) solutions proposées doivent donc avoir des qualités d'efficacité et de robustesse. L'efficacité est la qualité première d'une solution. Elle peut s'exprimer comme le gain maximum en temps de calcul sur un exemple donné (séquence particulière d'une EPS ou autre exemple). La robustesse d'une solution rend compte de sa "généricité". Une solution, qui améliore le temps de calcul d'une séquence mais qui, en contrepartie, dégrade la majorité des temps de calcul des autres séquences, est efficace, mais non robuste. Elle ne peut être retenue dans un contexte presse-bouton. Bien entendu, une solution qui permettrait de traiter toutes les séquences, mais qui dégraderait légèrement celles qui sont normalement calculées rapidement, peut être envisagée.

Les techniques de réécriture [LGTL85, Wil85, CY85, HK95] partent d'une formule booléenne *f*, la modifient, et génèrent une formule booléenne *g* de telle sorte que *f* et *g* soient équivalentes, c'est-à-dire que *Mintermes(f) = Mintermes(g)*. La réécriture, dans notre cas, vise à améliorer le temps de calcul du BDD de la formule. Dans d'autres domaines de recherche, la réécriture est utilisée afin de prouver l'équivalence entre deux formules booléennes.
De nombreuses réécritures ont été implémentées et testées afin de juger de leur efficacité et de leur robustesse. Elles se répartissent en plusieurs catégories développées dans les sections qui suivent.

6.1 SIMPLIFICATION DE LA FORMULE

La première catégorie de réécriture cherche à simplifier la formule. Cette simplification a deux buts principaux : supprimer les branches inutiles en remontant au plus haut les constantes booléennes, et préparer les formules à des réécritures plus importantes.

6.1.1 SIMPLIFICATION LOCALE D'UNE FORMULE

La simplification d'une formule booléenne consiste à appliquer l'ensemble des lois de simplification de l'algèbre de Boole. Ces lois sont rappelées ci-dessous.

Identité :

$$f + 0 \equiv f \qquad f \vee 0 \equiv f \qquad [6\text{-}1]$$
$$f + 1 \equiv 1 \qquad f \vee 1 \equiv 1$$
$$f \,.\, 0 \equiv 0 \qquad f \wedge 0 \equiv 0$$
$$f \,.\, 1 \equiv f \qquad f \wedge 1 \equiv f$$

Idempotence :

$$f + f \equiv f \qquad f \vee f \equiv f \qquad [6\text{-}2]$$
$$f \,.\, f \equiv f \qquad f \wedge f \equiv f$$

Absorption :

$$f + (f \,.\, g) \equiv f \qquad f \vee (f \wedge g) \equiv f \qquad [6\text{-}3]$$
$$f \,.\, (f + g) \equiv f \qquad f \wedge (f \vee g) \equiv f$$

Complétion :

$$f + \overline{f} \equiv 1 \qquad f \vee \neg f \equiv 1 \qquad [6\text{-}4]$$

$$f \,.\, \overline{f} \equiv 0 \qquad f \wedge \neg f \equiv 0$$

$$\overline{\left(\overline{f}\right)} \equiv f \qquad \neg(\neg f) \equiv f$$

Ces règles de réécriture modifient localement la formule étudiée en supprimant toutes les incohérences de construction. Elles sont le prérequis des autres règles de réécriture car elles permettent ensuite de manipuler des formules booléennes sur lesquelles il n'est plus utile de vérifier les cas "extravagants" du type *f+g+h+f*.

Parmi les simplifications locales, les lois d'identité jouent un rôle particulier, à plus d'un titre. Nous avons vu au paragraphe 5.2 que les arbres d'événements, qui utilisent les techniques de condition aux limites, génèrent des formules booléennes ayant de nombreux événements de configuration positionnés à vrai (=1) ou à faux (=0). Dans ce cas, les lois d'identité, et particulièrement les lois concernant l'élément absorbant des opérateurs booléens, permettent de diminuer de manière significative la taille des formules booléennes.
Considérons la formule suivante : $f + (g \,.\, cfg)$ avec $cfg = 0$ et f et g deux formules booléennes de taille quelconque.

- Si on construit directement le BDD de cette formule, on va d'abord construire le BDD de *f*, celui de *g*, puis celui de *cfg*. et seulement après le BDD de (*g . cfg*). Les règles de construction des BDD vont réaliser la simplification directement, mais le BDD de *g* sera forcément construit alors qu'il n'est pas utile.
- Si on simplifie la formule avant la construction du BDD, on ne construira que le BDD de *f*.

Le gain de cette réécriture est d'autant plus important que la formule *g* est grande.

Ce type de réécriture, même s'il est élémentaire, permet, dans le cas d'arbres d'événements utilisant la propagation d'événements de configuration comme l'EPS2, de pouvoir traiter beaucoup plus de séquences en moins de 10 minutes, comme le montre le tableau suivant.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819

6.1.2 PSEUDO-EVENEMENTS DE CONFIGURATION

Certaines méthodologies de construction imposent de représenter au sein des arbres de défaillance ou d'événements l'ensemble des hypothèses de l'analyse. Pour cette raison, il n'est pas exclu de trouver des événements de base qui ont une probabilité nulle. Ces événements sont tout de même conservés dans l'arbre à des fins qualitatives, afin de montrer qu'ils ont bien été pris en compte. Le fait de rendre leur probabilité nulle doit bien sûr être justifié par l'analyste.
A contrario, il est également possible de trouver des événements qui sont toujours réalisés. On les rencontre sous la forme d'événements de base ayant une loi de probabilité constante égale à 1. Ce type d'événements peut être associé à un sous-système afin de déterminer les coupes minimales dans lesquelles ce sous-système intervient.
Quel que soit l'avis que l'on porte sur ces méthodologies de construction, on doit prendre acte qu'elles sont utilisées dans le milieu industriel. Parallèlement à ces EPS, l'auteur du présent document a eu à traiter des arbres de défaillance issus de l'aéronautique ou du ferroviaire et qui utilisaient également ces pseudo-événements de configuration.
Les événements de base ayant des probabilités constantes, égales à zéro ou à un peuvent être considérés comme des pseudo-événements de configuration. La règle de réécriture envisagée consiste à détecter ce type de variables élémentaires est à les remplacer par les constantes booléennes 0 ou 1.
Si cette réécriture a permis de traiter des arbres de défaillance de grande taille provenant d'autres domaines industriels que le nucléaire, elle n'a pas engendré de gain conséquent pour l'EPS2.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Pseudo-CfgEvt	1696	80	2	5	0	0	8	28	1819

Nbr Réécriture	=0	<=10	<=20	<=30	<=40	<=50	<=60	>60	Tot
Pseudo-Even^{mnt}	177	219	132	235	253	417	186	200	1819

Ce deuxième tableau montre l'intérêt de cette réécriture. Pour chaque séquence, on a noté le nombre de fois qu'il a été possible d'appliquer cette règle de réécriture. Le tableau représente la répartition de toutes les séquences en fonction du nombre de réécriture de type "pseudo événements de configuration" possible. Au total, 200 séquences de l'EPS2 sont définies par plus de 60 événements de base ayant une probabilité nulle ou égale à un.

Il est à noter que cette technique peut être utilisée de manière systématique lorsque les résultats souhaités par l'utilisateur sont uniquement quantitatifs. Les événements de base toujours réalisés qui apportent des informations qualitatives, ne doivent pas être remplacés par des événements de configuration lors d'un calcul de coupes minimales.

6.1.3 COALESCENCE

La coalescence des formules cherche à "aplatir" les connecteurs commutatifs et associatifs.

Pour toute formule $f = g_1 \bullet ... \bullet g_i \bullet ... \bullet g_n$ [6-5]
où $g_i = h_1 \bullet ... \bullet h_k$
et $\bullet$ correspond à l'un des deux opérateurs $\vee$ $\wedge$
redéfinir f tel que $f = g_1 \bullet ... \bullet h_1 \bullet ... \bullet h_k \bullet ... \bullet g_n$

Cette réécriture est un prérequis important pour factoriser les équations ou pour trouver des facteurs communs à l'ensemble des formules. En revanche, elle peut très bien masquer des modules (Cf. chap.6.3 p.86) . Elle doit donc être utilisée en fonction des objectifs globaux du processus de réécriture.
Cette technique ne modifiant pas l'ordre des variables, nous pourrions penser, à tort, que la construction du BDD n'est pas influencée par cette réécriture. Si la taille finale du BDD est la même dans les deux cas (avec ou sans réécriture). L'ordre de construction des BDD va être légèrement différent, ce qui va modifier la taille des BDD intermédiaires et jouer un rôle dans la rapidité des calculs.
Nous avons vérifié expérimentalement sur l'EPS2, que cette règle de réécriture, appliquée seule, dégrade en général les performances des BDD.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Coalescence	1487	199	62	1	6	0	8	56	1819

Nbr Réécriture	=0	<=10	<=100	<=200	<=400	<=600	<=800	>800	Tot
Coalescence	195	132	203	211	559	331	140	48	1819

Ce second tableau représente la répartition de toutes les séquences en fonction du nombre d'opérandes déplacés lors de la réécriture de type "coalescence".

6.2 MINIMISATION DU NOMBRE D'OPERATEURS

Cette seconde catégorie de règles de réécriture regroupe les réécritures non locales qui vont chercher à minimiser le nombre d'opérateurs d'une formule. Minimiser le nombre d'opérateurs d'une formule permet de diminuer la taille des BDD intermédiaires.
Les réécritures vues précédemment favorisent, dans bien des cas, celles présentées dans ce paragraphe.

6.2.1 PROPAGATION DES CONSTANTES

La propagation des constantes est un exemple couramment employé pour expliquer la réécriture de formule booléenne [Nik00]. Elle consiste à appliquer les lois d'absorption d'une manière généralisée.
Cette règle de réécriture peut s'écrire de la manière suivante :

Pour toute formule $f = x \bullet g_1 \bullet ... \bullet g_n$ [6-6]
où x est une variable terminale
et $\bullet$ correspond à une des deux opérateurs $\vee$ $\wedge$
redéfinir f tel que $f = x \bullet g_1' \bullet ... \bullet g_n'$
avec $g_i' = g_i \mid x=1$ si $\bullet$ est l'opérateur "Et" : $\wedge$
ou $g_i' = g_i \mid x=0$ si $\bullet$ est l'opérateur "Ou" : $\vee$

Appliquer cette règle à une formule *f* modifie les sous-formules g_i de telle sorte qu'elles considèrent *x* comme une constante. A partir du moment où *x* est un opérande d'une formule *f*, il est propagé, en tant que constante, aux autres sous-formules de *f*.
Il est évident que cette réécriture ne doit être appelée que si au moins une sous-formule g_i est fonction de *x*. Dans le cas contraire, elle ne modifie en rien *f*.
Considérons la formule *f* suivante :

$$f = x \wedge g$$
$$g = z \vee h$$
$$h = x \wedge y$$

Après avoir appliqué la propagation des constantes, puis les règles de coalescence, la formule *f* est devenue :

$$f = x \wedge g$$
$$g = z \vee y$$

Cette réécriture n'est applicable à aucune des séquences des deux cas tests considérés EPS1 et EPS2. Dans un contexte industriel, il convient tout de même de la mettre en œuvre systématiquement, car elle ne peut qu'améliorer le temps de traitement de la formule, contrairement à d'autres réécritures qui peuvent dégrader les performances du calcul des BDD.

6.2.2 FACTEURS COMMUNS

La recherche de facteurs communs peut permettre une diminution du nombre d'opérations. Cette réécriture s'applique à un ensemble de formules qui sont définies à l'aide d'un même opérateur associatif-commutatif. Ces formules n'ont pas forcément des liens de parenté directe les unes avec les autres.
L'ensemble de formules suivant partagent des sous-formules qui peuvent être mises en facteur.

$$f_1 = g_1 \wedge g_2 \wedge g_3 \wedge g_4$$
$$f_2 = g_1 \wedge g_2 \wedge g_3 \wedge g_5$$
$$f_3 = g_3 \wedge g_4 \wedge g_6$$
$$f_4 = g_1 \wedge g_2 \wedge g_7$$

Nikolskaia a proposé dans [Nik00] un algorithme cherchant des facteurs communs d'une longueur minimale pour un ensemble de formules donné. Il se réalise en deux phases :

1. Dans un premier temps, il identifie les facteurs communs d'une longueur minimale *k*. Le résultat de cette première phase est l'ensemble *FP* des couples $\{\phi ; \pi\}$ où ϕ est un ensemble de sous-formules communes à plusieurs formules d'une taille minimale *k* et π l'ensemble des formules ayant en commun les éléments de ϕ.
   L'exemple précédent donne pour *k*=2 $<\{g_1 ; g_2 ; g_3\} ; \{f_1 ; f_2\}>$, $<\{g_1 ; g_2\} ; \{f_1 ; f_2 ; f_4\}>$, $<\{g_3 ; g_4\} ; \{f_1 ; f_3\}>$
2. Tant que *FP* n'est pas vide, il choisit dans *FP* un couple avec lequel il applique la mise en facteur commun. *FP* est remis à jour en prenant en compte ce choix. Ce choix peut être fait de plusieurs manières. L'heuristique testée consiste à choisir en fonction de la taille de π (ordre décroissant), puis de celle de ϕ (ordre décroissant). Cette opération pour l'exemple précédent est de choisir d'abord $<\{g_1 ; g_2\} ; \{f_1 ; f_2 ; f_4\}>$. L'ensemble *FP* devient alors $\{<\{g_3\} ; \{f_1 ; f_2\}> ; <\{g_3 ; g_4\} ; \{f_1 ; f_3\}>\}$. La taille de ϕ du premier couple est inférieure à *k* (=2). Il est supprimé et la mise en facteur est donc appliquée pour $<\{g_3 ; g_4\} ; \{f_1 ; f_3\}>$.

Le résultat de cette réécriture est alors :

$$f_1 = h_1 \wedge h_2$$
$$f_2 = h_1 \wedge g_3 \wedge g_5$$
$$f_3 = h_2 \wedge g_6$$
$$f_4 = h_1 \wedge g_7$$
$$h_1 = g_1 \wedge g_2$$
$$h_2 = g_3 \wedge g_4$$

On peut alors constater que le nombre d'opérations binaires est passé de 10 à 7.

Pour qu'elle fournisse un résultat optimal, cette réécriture nécessite un minimum de prérequis. Avant d'appeler cette réécriture, il convient de faire toutes les simplifications locales possibles et d'utiliser les règles de coalescence pour essayer de maximiser les chances de trouver des facteurs communs.

Cet algorithme a été implémenté au sein de Hévéa. Dans le cas de l'EPS2, il donne les résultats figurant au tableau ci-dessous. Après avoir appliqué dans l'ordre la coalescence et les simplifications locales, la recherche de facteurs communs a été appliquée une première fois en fixant *k* à 6, afin de faire ressortir en priorité les facteurs communs de grande taille, puis une seconde fois pour *k* égal à 2.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Facteur commun	1609	146	27	8	2	0	4	23	1819

Nbr Réécriture	=0	<=20	<=40	<=60	<=80	<=100	<=120	>120	Tot
Facteur commun	198	474	617	161	202	109	58	0	1819

La seconde partie du tableau représente la répartition de toutes les séquences en fonction du nombre de facteurs communs trouvés par cette réécriture. Dans l'exemple donné précédemment, deux facteurs communs ont été trouvés ($g_1 \wedge g_2$ et $g_3 \wedge g_4$).
Cette réécriture dégrade légèrement les séquences qui sont normalement calculées rapidement. Mais 13 des 40 séquences posant un problème ont pu être calculées à l'aide de cette dernière.

6.2.3 FACTORISATION

La factorisation s'applique à toute formule qui est une disjonction de conjonctions (ou inversement une conjonction de disjonctions) dont les sous-formules ont des parties communes. Il est possible de factoriser les formules suivantes qui ne sont autres que les formules du paragraphe précédent auxquelles on a ajouté *f*, la disjonction des f_i :

$$f = f_1 \vee f_2 \vee f_3 \vee f_4$$
$$f_1 = g_1 \wedge g_2 \wedge g_3 \wedge g_4$$
$$f_2 = g_1 \wedge g_2 \wedge g_3 \wedge g_5$$
$$f_3 = g_3 \wedge g_4 \wedge g_6$$
$$f_4 = g_1 \wedge g_2 \wedge g_7$$

Cette règle de réécriture a facilité l'étude du système d'arrêt d'urgence d'un réacteur nucléaire [CDLR98].
Dans les EPS, le traitement d'un ensemble de séquences peut entraîner l'utilisation massive de ce type de réécritures. Il existe des cas pratiques où un calcul de probabilité d'occurrence d'une conséquence ne peut pas se borner à l'addition des probabilités d'occurrence des séquences y aboutissant. Il convient alors de traiter ce calcul en considérant l'ensemble des séquences aboutissant à cette conséquence de manière logique. De plus, il n'est pas rare d'utiliser un ensemble de séquences comme initiateur d'un autre arbre d'événements. Dans ces deux cas, on doit alors traiter une formule qui est une disjonction de conjonctions.
Dans la suite du paragraphe, nous considérerons uniquement une formule sous une forme de somme de produits, le traitement des produits de sommes étant dual.

Une somme de produits peut donc s'écrire sous la forme suivante :

$$F = \bigcup_{i=0}^{i<n} P_i \qquad avec\ P_i = \bigcap_{j=0}^{j<m_i} R_j \qquad [6\text{-}7]$$

où les R_j sont des formules booléennes

Précisons dès à présent, que le but de cette réécriture n'est pas de rendre les produits disjoints [CDRB99], mais de réduire le nombre de calculs intermédiaires et donc le nombre d'opérateurs.

Soit *S* l'ensemble des produit P_0, ..., P_n. Une manière de factoriser S est de choisir une variable *X* parmi les R_j des P_i. *X* peut apparaître positivement ou négativement dans les P_i. *S* peut être décomposé en trois sous-ensembles disjoints, *U*, *V* et *W* :

- *U* : l'ensemble des P_i contenant *X* positivement,
- *V* : l'ensemble des P_i contenant *X* négativement,
- *W* : l'ensemble des P_i ne contenant pas *X*.

La formule *F* peut alors se récrire :

$$F = \left(X \wedge U_{X \leftarrow 1}\right) \vee \left(\neg X \wedge V_{X \leftarrow 0}\right) \vee W \qquad [6\text{-}8]$$

On poursuit la factorisation de manière récursive sur les formules $U_{X \leftarrow 1}$, $V_{X \leftarrow 0}$ et *W*.

En pratique, l'ordre de factorisation est primordial. Il existe de nombreuses heuristiques fixant cet ordre. Seule l'expérimentation peut permettre de déterminer l'heuristique la plus robuste.
Les heuristiques peuvent être décomposées en deux familles distinctes :

- Heuristique statique : L'ordre des variables est choisi une fois pour toutes avant le lancement de la factorisation.
- Heuristique dynamique : A chaque étape de la factorisation, on choisit la variable à factoriser en fonction de l'ensemble de produits en cours.

Nous avons considéré les heuristiques suivantes :

1. Heuristique manuelle (Statique) : Cette heuristique prend en paramètre un ordre de variable sous la forme d'un sélecteur de variables. Cela permet de tester des heuristiques statiques sans les programmer au sein de l'interface. Cela a permis aussi de vérifier l'algorithme de factorisation.
2. Heuristique aléatoire (Statique) : Cette heuristique fixe aléatoirement l'ordre des variables. Elle a été développée à des fins de test principalement.
3. Heuristique naturelle (Statique) : Chaque séquence du sélecteur définit un ordre naturel. Cette heuristique essaye de définir un ordre global sur l'ensemble des séquences. Pour cela, on définit un graphe orienté acyclique à l'aide des séquences du sélecteur en rejetant les transitions qui rendraient le graphe cyclique et on effectue un tri topologique sur ce graphe.
4. Heuristique d'occurrence (Statique) : L'ordre des variables se fait en fonction du nombre de séquences utilisant cette variable (occurrence des variables). L'idée est de mettre en avant les variables les plus souvent utilisées.
5. Heuristique d'occurrence avec modificateur pour les littéraux purs (Statique) : On appelle littéral pur une variable n'apparaissant que positivement ou que négativement. Dans le principe de factorisation, elles ont un rôle particulier. Si X est une variable n'apparaissant que positivement (resp. négativement) alors l'ensemble V (resp. U) est vide. On a donc souhaité tester les importances dans le choix de l'ordre. Pour cela, cette heuristique prend en paramètre un coefficient réel P supérieur à 1. L'ordre est alors déterminé en fonction du nombre d'occurrences de chaque variable, multiplié éventuellement par P, si la variable est un littéral pur.
6. Heuristique des plus petits produits (Dynamique) : A chaque étape, on choisit la variable X qui maximise Weight[X]. Soit S' l'ensemble des produits P_0, ..., P_m, de l'ensemble courant qui contiennent soit +X, soit -X. L'opérateur Size(P_i) retourne le nombre de variables de P_i. On détermine Weight[X] de la manière suivante :

$$\text{Weight}[X] = \sum_{i=0}^{m} \left(\frac{1}{Beta^{Size(Pi)}} \right) \qquad [6\text{-}9]$$

où Beta est un nombre réel supérieur à 1 fixé expérimentalement.
L'idée est de proposer la variable qui factorise le plus de petit produit. Cette heuristique est dérivée de travaux sur la résolution du problème SAT [DABC96].

Afin de déterminer l'efficacité des différentes heuristiques, nous avons testé la factorisation en l'appliquant à l'EPS1, pour deux sélecteurs de séquences différents :

1. Le premier, correspondant aux séquences aboutissant à la fusion du cœur sous haute pression, fait intervenir 55 séquences. Pour cette ensemble de séquences, l'heuristique la plus efficace a permis de diminuer le temps de calcul d'un facteur 58 et la taille du BDD final d'un facteur 3,47. Des résultats plus détaillés sont présentés dans la suite de ce paragraphe.
2. Le second correspond à un ensemble de 135 séquences qui engendrent la perte d'un réacteur nucléaire, lorsque la pression du cœur est basse. Dans ce cas, le calcul du BDD n'a pas pu être réalisé en moins d'une heure quelle que soit l'heuristique de factorisation. L'efficacité des heuristiques a donc été testée sur un calcul de p-BDD d'ordre 3 (puis d'ordre 4 pour les heuristiques les plus intéressantes).

Ces expérimentations ont mis en évidence l'importance de la factorisation. Pour toutes les heuristiques raisonnables, nous avons obtenu un gain significatif en termes de temps de calcul, de taille du BDD final et de nombre de nœuds intermédiaires créés. Ces résultats sont valables aussi bien pour un calcul de BDD que pour un calcul de p-BDD.

Le Tableau 6-1 montre les gains obtenus pour le premier ensemble de séquences en fonction des différentes heuristiques testées. Contrairement aux autres expérimentations de ce chapitre, ces résultats ont été obtenus sur une machine SUN Sparc4 à 255 Mo de mémoire vive avec un processeur 133 MHz. La limite concernant le nombre de nœuds qu'il est possible de créer a été portée à 10 millions, afin de pouvoir faire ces tests sans être bloqué par un problème de mémoire.

Heuristique		Nombre	Temps	Taille du	Nombre de	Temps (sec.)		Gain			
Nom	Par.	Opérateur	Génération	BDD	Noeuds	Utilisateur	Système	BDD	Noeuds	Temps	Moyenne
Référence	-	477	0.26	840 016	6 512 433	663.96	2.74	1.00	1.00	1.00	1.00
Naturel	-	211	0.77	1 013 825	2 888 192	209.08	0.94	0.83	2.25	3.17	2.09
Occurrence	-	203	0.72	439 122	1 217 865	43.80	0.46	1.91	5.35	15.06	7.44
Occurrence+	1.25	206	0.68	327 160	1 399 671	34.74	0.51	2.57	4.65	18.91	8.71
Occurrence+	1.5	199	0.65	365 407	1 374 580	32.58	0.42	2.30	4.74	20.20	9.08
Occurrence+	2	195	0.64	365 767	1 417 212	33.22	0.53	2.30	4.60	19.75	8.88
Occurrence+	3	183	0.64	585 267	1 852 798	57.42	0.55	1.44	3.51	11.50	5.48
Occurrence+	4	189	0.62	882 119	2 873 835	459.08	0.96	0.95	2.27	1.45	1.56
Petits Produits	1.25	203	0.62	439 122	1 212 843	44.15	0.40	1.91	5.37	14.97	7.42
Petits Produits	1.5	199	0.48	305 516	1 026 507	23.80	0.41	2.75	6.34	27.54	12.21
Petits Produits	2	224	0.49	279 060	863 671	18.41	0.26	3.01	7.54	35.71	15.42
Petits Produits	3	211	0.45	242 068	784 678	11.16	0.24	3.47	8.30	58.48	23.42
Petits Produits	4	210	0.44	239 090	911 005	14.02	0.31	3.51	7.15	46.52	19.06

Tableau 6-1 : Factorisation sur un ensemble de 55 séquences de l'EPS1 : calcul du BDD

L'heuristique "Référence" correspond à la formule prise dans l'état initial sans factorisation. Certaines heuristiques prennent un paramètre (colonne "Par.") jouant un rôle non négligeable sur son efficacité. Le nombre d'opérateurs correspond pour l'heuristique "Référence" au nombre d'opérations utilisé pour décrire la somme de produits et pour les autres heuristiques au nombre d'opérations pour effectuer cette même somme, mais après factorisation. Le temps de génération, qui correspond au temps nécessaire pour réaliser la factorisation de la formule, est négligeable par rapport au gain que cette réécriture apporte.
Les métriques de comparaison utilisées sont :

- Le temps CPU utilisé pour calculer le BDD de la formule (somme des temps utilisateur et système),
- La taille du BDD final codant la formule,
- Le nombre de nœuds créés pour construire le BDD final,

L'heuristique des plus petits produits est actuellement la plus performante. Cette expérimentation a montré qu'avec un béta supérieur à 2, les résultats étaient plus qu'intéressants.

Pour information, les tableaux suivants rendent compte des expérimentations réalisées sur le second ensemble de séquences pour un calcul de p-BDD d'ordre 3 puis d'ordre 4.

Heuristique		Nombre	Temps	Taille du	Nombre de	Temps (sec.)		Gain			
Nom	Par.	Opérateur	Génération	BDD	Noeuds	Utilisateur	Système	BDD	Noeuds	Temps	Moyenne
Référence	-	1342	0.35	1 162	368 468	348.41	0.13	1.00	1.00	1.00	1.00
Naturel	-	419	2.75	771	338 188	767.81	0.17	1.51	1.09	0.45	1.02
Occurrence	-	587	6.37	1 007	117 265	155.34	0.11	1.15	3.14	2.24	2.18
Occurrence+	1.25	583	6.18	885	123 517	164.67	0.08	1.31	2.98	2.12	2.14
Occurrence+	1.5	557	5.93	881	143 824	167.26	0.03	1.32	2.56	2.08	1.99
Occurrence+	2	546	5.21	881	138 392	62.79	0.04	1.32	2.66	5.55	3.18
Occurrence+	3	543	5.05	921	175 564	64.12	0.15	1.26	2.10	5.42	2.93
Occurrence+	4	530	4.57	1 043	282 716	102.75	0.19	1.11	1.30	3.39	1.93
Petits Produits	1.25	582	5.08	1 007	121 320	150.67	0.05	1.15	3.04	2.31	2.17
Petits Produits	1.5	537	2.43	1 043	132 700	145.64	0.08	1.11	2.78	2.39	2.09
Petits Produits	2	516	1.92	1 060	90 123	53.12	0.07	1.10	4.09	6.55	3.91
Petits Produits	3	526	1.84	1 027	98 716	51.56	0.02	1.13	3.73	6.76	3.87
Petits Produits	4	532	1.84	1 027	92 754	48.74	0.06	1.13	3.97	7.14	4.08

Tableau 6-2 : Factorisation sur un ensemble de 135 séquences de l'EPS1 : calcul du p-BDD à l'ordre 3

Heuristique		Taille du	Nombre de	Temps (sec.)		Gain			
Nom	Par.	BDD	Noeuds	Utilisateur	Système	BDD	Noeuds	Temps	Moyenne
Référence	-	14 035	1 313 153	12935.45	0.37	1.00	1.00	1.00	1.00
Occurrence+	2	13 384	468 566	1381.62	0.18	1.05	2.80	9.36	4.40
Occurrence+	3	15 481	615 466	1493.78	0.28	0.91	2.13	8.66	3.90
Petits Produits	2	17 490	256 334	1153.68	0.25	0.80	5.12	11.21	5.71
Petits Produits	3	12 518	301 775	1116.15	0.12	1.12	4.35	11.59	5.69
Petits Produits	4	12 473	268 288	1028.97	0.08	1.13	4.89	12.57	6.20

Tableau 6-3 : Factorisation sur un ensemble de 135 séquences de l'EPS1 : calcul du p-BDD à l'ordre 4

En plus de son application à la factorisation d'un ensemble de séquences, il n'est pas improbable de pouvoir utiliser cette réécriture directement au sein d'une formule booléenne quelconque. Cette réécriture a donc été

appliquée à l'ensemble des séquences de l'EPS2. Les prérequis permettant de mettre en œuvre la factorisation sont les mêmes que ceux de la recherche de facteurs communs. Ils consistent à appliquer dans l'ordre la coalescence et les simplifications locales.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Factorisation	1615	153	16	8	1	0	0	26	1819

Nbr Réécriture	=0	=1	=2	=3	<=5	<=7	<=9	=10	Tot
Factorisation	514	656	120	112	86	220	81	30	1819

Le second tableau représente la répartition de toutes les séquences en fonction du nombre de fois que cette réécriture a pu être mise en œuvre. C'est-à-dire le nombre de fois où une factorisation était possible.
Cette réécriture est celle qui donne actuellement les meilleurs résultats. Les temps de calcul des séquences ne posant pas de problèmes ne sont pas trop dégradés (principe de robustesse). De plus 14 des 40 séquences difficiles sont calculables à l'aide de cette réécriture.

6.3 RECHERCHE DE MODULES

Dans la littérature [BE65, Cha75, Loc81, STK88, Yll88, KHI89] sur les arbres de défaillance, une des simplifications les plus utilisées consiste à isoler les modules trouvés dans les arbres. Un module est une sous-formule ne partageant pas de variables avec le reste de l'arbre. Il peut alors être évalué indépendamment de l'ensemble de l'arbre et être représenté au sein de l'arbre originel par un événement de base. La probabilité et les coupes minimales associées à l'arbre prendront en compte les résultats de calcul des modules.
Il existe des algorithmes efficaces pour repérer les modules, si ceux-ci sont déjà présents dans l'arbre [DR96]. Il est important de mentionner que dans la majorité des cas, les modules sont cachés et il est difficile de savoir quelles règles de réécriture appliquer pour les faire émerger. Par exemple, les deux formules $f = (x \vee g) \wedge (x \vee h)$ et $f' = x \vee (g \wedge h)$ sont équivalentes. La seconde est modulaire, alors que la première ne l'est pas.
Avec le codage des formules par les BDDs, le gain obtenu grâce à ce type de détection est assez faible (même s'il est toujours bon à prendre). De plus, intégrer le traitement de formules booléennes modulaires au sein d'un logiciel comme Aralia est une tâche posant de nombreux problèmes d'ingénierie. Tous les algorithmes manipulant les BDD devraient être reconsidérés. Le gain en terme de temps de calcul doit être par conséquent important pour prendre la décision de l'implémenter.
Une expérimentation a été menée sur l'EPS2 afin de juger des gains qu'il serait possible d'obtenir à l'aide de la modularité. Elle a consisté pour chaque séquence, à chercher et à calculer indépendamment tous ces modules et de comparer le temps de calcul total avec le temps de calcul de référence.
Les résultats n'ont pas été suffisamment satisfaisants pour que cette approche soit généralisée.
Ces résultats s'expliquent par le fait qu'il n'y a que très peu de modules au sein des séquences, et que dans tous les cas, les plus gros modules rencontrés ont un nombre de variables terminales inférieur à 10 % du nombre de variables terminales de la formule de référence.

6.4 LES HEURISTIQUES D'ORDONNANCEMENT

Pour construire le BDD associé à une formule, on doit choisir un ordre total sur les variables de cette formule. Il est connu depuis le début de l'utilisation des BDDs que l'efficacité de la méthode dépend très largement de ce choix. Malheureusement, trouver un bon ordre est un problème difficile [FS90] et l'on est réduit à utiliser des heuristiques. De nombreuses heuristiques ont été proposées dans la littérature [BRKM91, FFK88, FFM93, FMK91]. Ces heuristiques sont fondées sur la topologie de la formule étudiée. Elles sont dites statiques, puisqu'elles choisissent un ordre une fois pour toutes.
Toutes ces heuristiques peuvent être simulées avec l'algorithme suivant. On associe à chaque formule des attributs qui dépendent des attributs de ses formules mères ou filles et de la nature de son opérateur. On modifie alors l'ordre des sous-formules en fonction de leurs attributs. L'ordre total sur les variables de la formule est ensuite obtenu en la parcourant en profondeur et à gauche.

Les expérimentations réalisées dans [BBR97, Nik00] partagent les heuristiques en deux catégories :

- Les heuristiques qui font des choix en propageant l'information de la racine de la formule vers ses feuilles. L'efficacité de ces heuristiques dépend de la façon dont la formule est écrite. Elles ne sont donc pas robustes.
- Les heuristiques qui associent des poids ou des notes aux feuilles de la formule et qui propagent cette information vers le sommet. Ces heuristiques sont robustes, car quelles que soient les permutations aléatoires des opérateurs associatifs-commutatifs, les ordres des variables en résultant sont toujours proches. Malheureusement, ces heuristiques ne sont pas en général efficaces.

Pour cette raison, la référence [BBR97] propose d'essayer plusieurs réécritures pseudo-aléatoires des formules et d'appliquer une heuristique de la première catégorie sur chacune d'entre elles.
Une autre approche a été proposée dans le cadre de l'analyse de circuits. Il s'agit des techniques de réordonnancement dynamique qui changent l'ordre des variables en cours de construction du BDD [Rud93, FYBS93]. Cette technique peut être appliquée dans le cadre des arbres d'événements. Comme [Nik00] le suggère, il faudrait sans doute trouver des stratégies de réordonnancement adaptées à ce type de formules.

6.4.1 OPTIMISATION DU REORDONNANCEMENT

Les techniques classiques d'ordonnancement consistent à définir un ordre en fonction d'un tri sur les attributs associées aux formules. Nous proposons de chercher un ordre sur les sous-formules qui optimise une grandeur dépendant des caractéristiques des ses sous-formules.

Le but de cette approche est d'essayer de mettre côte-à-côte les sous-formules de F partageant beaucoup de feuilles. Pour cela, on définit un certain nombre de métriques pour chaque sous-formule f_i qui sont :

- N_i : le nombre de feuilles de f_i.
- L_i : le nombre de feuilles n'appartenant qu'à f_i.
- P_{ik} : le nombre de feuilles partagées entre f_i et f_k.

Soit V, un vecteur $v_1, ..., v_n$ représentant l'ordre des sous-formules de F. $v_i = k$ signifie que la sous-formule f_k est positionnée à la $i^{ème}$ position.
On cherche V qui optimise une fonction-objectif W définie par

$$W(v_1, ..., v_n) = \mathrm{Fct}(Nv_i, Lv_i, Pv_iv_k) \qquad [6\text{-}10]$$

L'objectif étant de mettre côte-à-côte les sous-formules f_k partageant le plus grand nombre de feuilles, la fonction-objectif à maximiser peut être la suivante :

$$W\left(v_1,\ldots,v_n\right) = \sum_{i=1}^{n-1}\left(Pv_iv_{i+1}\right) \qquad [6\text{-}11]$$

Certaines sous-formules de F ne partagent que très peu de feuilles. Afin de ne pas perturber le calcul du BDD, il convient de les traiter soit au début, soit à la fin. Trouver un ordre adéquat qui prenne en compte ce second objectif peut se faire en maximisant la fonction-objectif suivante :

$$W\left(v_1,\ldots,v_n\right) = \sum_{i=1}^{n-1}\left[\left(1-\left(\frac{Lv_i}{Nv_i}\right)^{n-i}\right)\times Pv_iv_{i+1}\right] \qquad [6\text{-}12]$$

La seconde idée, qui nous paraît importante, est que seules les modifications sur les premiers niveaux de la formule ont une répercussion sur le calcul du BDD. Il n'est pas nécessaire d'appliquer cette heuristique à l'ensemble des sous-formules, seuls les N premiers niveaux sont à réordonner.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Optimisé	1600	191	4	4	0	0	0	20	1819

Nbr Réécriture	=0	=1	=2	=3	=4	=5	=6	=7	Tot
Optimisé	366	184	428	313	211	172	119	26	1819

Le second tableau représente la répartition de toutes les séquences en fonction du nombre de fois que cette réécriture a modifié l'ordre des sous-formules.
Bien que cette réécriture n'ait pas donné les résultats attendus, elle est celle qui donne les moins mauvais résultats puisqu'elle permet de traiter pratiquement toutes les séquences à l'exception d'une vingtaine d'entre elles.

6.4.2 REORDONNACEMENT ALEATOIRE

Pour aller plus loin que la proposition de [BBR97], la méthodologie pragmatique qui suit donne de bons résultats si on accepte d'en payer le prix en terme de temps de calcul. Elle consiste à allouer un temps de calcul maximum relativement court. Tant que l'on n'arrive pas à calculer le BDD de la formule, on réalise des permutations aléatoires des connecteurs commutatifs-associatifs sur la formule et on relance le calcul.

On peut décrire cette méthode à l'aide du pseudo-algorithme suivant :

```
01 Soit f une formule booléenne,
02      T un délai,
03      cpt un compteur,
04      ShuffleArgument(f) une procédure réalisant des permutations
05          aléatoires sur la fonction donnée en paramètre.
06 T = ΔT
07 cpt = 0
08 Tant que (cpt < max)
09      Si (cpt mod 10 = 0) Alors T += ΔT
10      Limiter la durée du prochain calcul de BDD à T
11      Shuffle-Argument(f)
12      Calculer le BDD de f
13      Si le calcul aboutit
14          Alors Sortir
15          Sinon cpt++
```

Cette méthode utilise deux paramètres : ΔT, qui correspond à un incrément de temps destiné à laisser un peu plus de temps pour le calcul au fur et à mesure des essais, et *max* qui fixe le nombre maximum d'essais que l'on autorise.

Cette technique permet de traiter la très grande majorité des séquences de l'EPS2 en moins de 10 minutes.
Les tests ont été faits en fixant *max* = 120 itérations et ΔT à 10 secondes. Le délai T est incrémenté de ΔT toutes les 20 itérations. Dans ce contexte, le temps que peut prendre le processus aléatoire pour trouver une solution peut être supérieur à 10 minutes. En revanche, si une solution est trouvée, son BDD est calculé en moins d'une minute.
La formule de départ provient de la réécriture ayant donné les meilleurs résultats, à savoir l'optimisation de réordonnancement. En fait seules les 24 séquences les plus difficiles de cette réécriture ont besoin de cette technique pour être évaluées. Comme le montre le second tableau, les autres se calculent en une itération, c'est-à-dire en moins de 10 secondes.

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Original	1202	149	3	14	0	0	249	202	1819
Simplifié	1653	112	12	2	0	0	4	36	1819
Optimisé	1600	191	4	4	0	0	0	20	1819
Processus	1528	262	5	8	11	2	3	-	1819
Aléatoire	1602	198	19	0	0	0	0	0	1819

Nbr d'itération	=1	<=10	<=20	<=30	<=40	<=80	<=120	>120	Tot
Aléatoire	1795	17	2	2	1	0	2	0	1819

La ligne "processus" du premier tableau montre que le temps de calcul a été supérieur à 10 minutes pour trois séquences. Il a fallu 38 itérations pour calculer une des trois séquences et 118 itérations pour les deux autres (soit un peu plus de 40 minutes). Dans tous les cas, une fois que l'on dispose de la formule réécrite, il faut moins de 20 secondes pour calculer son BDD (ligne "aléatoire" du premier tableau).

Il est à noter que cette méthode peut être améliorée sur de nombreux points. En effet, la permutation aléatoire est réalisée sans mémoire. On ne l'oriente pas en fonction des réussites et des échecs des précédentes itérations. Une des améliorations envisageables est de "réparer" localement les permutations aléatoires n'ayant aucune chance de donner des résultats optimaux. Si deux sous-formules partagent un nombre important de variables terminales (supérieur à un seuil fixé en pourcentage), il est nécessaire de les rapprocher l'une de l'autre pour avoir des résultats satisfaisants.

Dans le cas présenté, on cherche uniquement à calculer le BDD. Un autre objectif pourrait être de minimiser la taille du BDD jusqu'à un seuil déterminé, ou encore de considérer une combinaison des deux objectifs précédents consistant à retenir la formule ayant une taille minimum sur *n* itérations ou la première permutation y arrivant, si au bout de ces *n* itérations le BDD n'a pas pu être déterminé.

6.5 CAS PARTICULIER :

Le but des réécritures proposées dans cette section n'est pas de simplifier le BDD en vue de calculer des probabilités, mais de le simplifier afin de pouvoir calculer le ZBDD codant les coupes minimales de la formule considérée. Les BDD ainsi simplifiés peuvent être utilisés pour calculer une approximation de la probabilité de la formule considérée. Les techniques proposées sont similaires à celles utilisées par les logiciels de quantification des EPS. Rien ne garantit qu'elles puissent être utilisées sans précaution dans le cadre des arbres d'événements.
La seconde particularité de ces réécritures est qu'elles s'appliquent uniquement aux séquences d'arbres d'événements comportant des parties négatives, c'est-à-dire comportant des systèmes de sûreté ayant fonctionné.

Considérons pour la suite la séquence 2 de l'arbre d'événements de la Figure 6.1.

La formule booléenne associée à Seq2 est :

$Seq2 := \neg G \wedge F;$

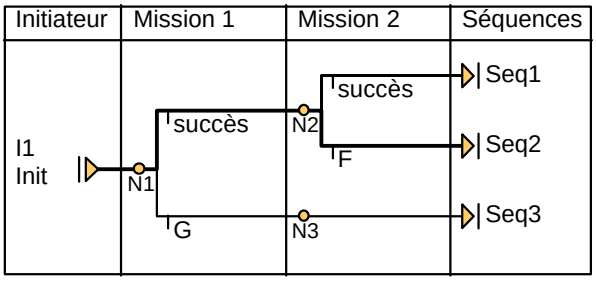


Figure 6.1 : Exemple d'arbre d'événements

Une première technique d'approximation consiste à ne pas tenir compte des parties négatives des séquences.
Cette approximation n'en est pas une si G est monotone et si G et F ne partagent aucune variable terminale (feuille(G) $\cap$ feuille(F) = {$\varnothing$}). Dans ce cas, nous pouvons écrire :

$$MCS(\neg G \wedge F) = MCS(F). \qquad [6\text{-}13]$$

Cette simplification a le mérite de la clarté et de l'efficacité. Mais, il est utopique de considérer qu'il n'existe pas de dépendances fonctionnelles entre les systèmes de sûreté.

Le principal problème de cette première technique est qu'elle ne prend pas en compte les composants qui sont supposés fonctionner dans les branches de succès des arbres de défaillance.
Cette réflexion a abouti à la seconde technique d'approximation qui consiste, toujours en considérant que G est monotone, à supprimer de la formule l'ensemble des variables terminales appartenant à G et n'appartenant pas à F, c'est-à-dire l'ensemble des variables terminales n'appartenant qu'à G. Ceci est réalisable à l'aide du quantificateur existentiel.
Soit E, l'ensemble des variables terminales de G n'appartenant pas à F. E = feuilles(G) \ feuilles(F).

$$MCS(\ exist(E, \neg G \wedge F)\ ) = MCS(\neg G \wedge F) \qquad [6\text{-}14]$$

On retrouve dans cette technique la définition des coupes minimales du chapitre 2, puisqu'on cherche à simplifier le BDD, codant une séquence, en supprimant les composants ayant fonctionné, c'est-à-dire les non-défaillances qui n'apportent que peu d'informations sur la séquence.
En fait cette réécriture permet de calculer l'enveloppe monotone de la séquence, si tous les systèmes de sûreté sont décrits par des fonctions monotones.

Une autre manière d'implémenter cette technique consiste à utiliser le quantificateur existentiel, non pas sur la formule codant l'ensemble de la séquence (comme c'est le cas dans l'équation ci-dessus), mais uniquement sur la partie succès de la séquence.

$$MCS(\ (exist(E, \neg G)) \wedge F\ ) = MCS(\neg G \wedge F) \qquad [6\text{-}15]$$

Cette implémentation a l'avantage de faire des simplifications tout au long de la construction des BDD intermédiaires et sans attendre que le BDD codant la séquence soit construit.

Le premier tableau regroupe les temps de calcul nécessaires pour obtenir les coupes minimales des séquences de la première EPS. La première ligne est le résultat de référence, obtenu sans réécriture de la formule. Les lignes

"Success-True" et "Success-Exist" sont relatives aux deux techniques présentées dans ce paragraphe. Le nombre entre parenthèses, figurant après la mention MOCUS, est le seuil relatif considéré pour ces résultats.
Le deuxième tableau présente les résultats relatifs aux calculs du BDD de la seconde EPS.
Le troisième tableau récapitule les expériences réalisées pour calculer les coupes minimales des séquences de la seconde EPS. La première ligne correspond, cette fois-ci, à une technique de type réordonnancement aléatoire utilisée pour le calcul des coupes minimales.

ZBDD(EPS1)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Référence	230	73	10	0	0	8	8	0	329
Success-True	228	78	11	0	8	0	4	0	329
Success-Exist	234	73	13	1	8	0	0	0	329
MOCUS (10^{-3})	282	41	5	1	0	0	0	0	329
MOCUS (10^{-4})	222	81	20	6	0	0	0	0	329
MOCUS (10^{-5})	186	80	51	6	6	0	0	0	329

BDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Référence	1202	149	3	14	0	0	249	202	1819
Success-True	1814	3	2	0	0	0	0	0	1819
Success-Exist	1689	120	2	6	2	0	0	0	1819

ZBDD(EPS2)	<1s	<5s	<20s	<1m	<5m	<10m	>10m	Mem	Total
Aléatoire	1333	340	124	6	2	0	0	14	1819
Success-True	1520	201	71	6	9	4	8	0	1819
Success-Exist	1503	157	72	16	35	10	26	0	1819
MOCUS (10^{-3})	1697	90	23	9	0	0	0	0	1819
MOCUS (10^{-4})	1641	114	39	20	5	0	0	0	1819
MOCUS (10^{-5})	1591	100	83	25	20	0	0	0	1819

Pour les séquences de taille moindre (EPS1), les temps de calcul induits par ces différentes approches sont relativement proches. Les temps de calcul obtenus à l'aide du quantificateur existentiel se situent entre les temps de calcul de MOCUS de seuils relatifs 10^{-4} et 10^{-5}. Curieusement les résultats obtenus en ne considérant pas les parties négatives, sont moins bons que ceux utilisant le quantificateur existentiel, alors que la suppression est plus importante. Cela peut s'expliquer par le fait que la non-prise en compte des parties négatives va modifier l'ordonnancement des variables terminales et engendrer alors une mauvaise heuristique d'ordonnancement.

Les BDD des formules engendrées par les deux méthodes sont calculés sans aucune difficulté. Il est donc possible de calculer les approximations réalisées par ces méthodes pour toutes les séquences. Le calcul des coupes minimales posent en revanche plus de problèmes. Les coupes minimales de 26 des 1819 séquences de l'EPS ne sont pas calculables en moins de 10 minutes, à l'aide de la réécriture utilisant le quantificateur existentiel. Ce nombre est ramené à 8, si nous supprimons les parties négatives des séquences. Il est à noter que ces résultats sont déjà appréciables, au vu des résultats du réordonnancement aléatoire appliqué pour le calcul des coupes minimales sur ces séquences. En effet, dans ce cas, 14 séquences n'ont pas abouti. Par conséquent, lorsque l'on souhaite calculer les coupes prépondérantes, il est préférable d'utiliser l'algorithme MOCUS qui a d'ailleurs été conçu avec cet objectif précis.

6.6 CONCLUSION

Les techniques de réécritures sont nombreuses et permettent des avancées dans le traitement des grosses formules booléennes. En général, les formules booléennes qui résultent d'une analyse manuelle d'un système sont simplifiables. L'analyste ne cherche pas à produire une formule booléenne réduite, mais à décrire le fonctionnement (ou le dysfonctionnement) de son système. Les techniques qui visent à simplifier les formules booléennes sont donc systématiquement à retenir. Citons, pour rappel, toutes les lois de simplification locale et principalement celles qui concernent les éléments absorbants des opérateurs.
La réécriture de type coalescence pose un double problème. Elle permet, en général, de mettre en œuvre les autres réécritures (factorisation, recherche de facteurs communs, simplification locale, ...) et, en cela, est utile. En revanche, appliquée de manière systématique, elle dégrade les performances de calcul. Bien que cela n'ait pas été mis en œuvre dans cette étude, il conviendrait peut-être de n'appliquer la coalescence que pour certaines variables et uniquement dans un but précis défini à l'avance. Tous les tests, permettant de savoir si une réécriture (factorisation, ...) est possible, devraient pouvoir être réalisés en prenant en compte les coalescences possibles. Il

conviendrait alors d'appliquer la coalescence uniquement sur les variables impliquées dans la réécriture considérée.

L'idée d'optimisation du réordonnancement permet d'ouvrir une nouvelle voie de réflexion dans les techniques d'heuristiques et doit pouvoir être améliorée. Le réordonnancement aléatoire est, quant à lui, une technique d'ingénierie assez générale qui offre de bonnes qualités de robustesse. Il est également possible de l'améliorer afin d'éviter les permutations n'ayant aucune chance de donner le résultat rapidement.

Pour finir, l'utilisation d'un quantificateur existentiel sur les branches de succès d'une séquence permet de calculer directement le BDD codant l'enveloppe monotone de toutes les séquences des EPS étudiées. Le temps de calcul est en général inférieur à celui du calcul des coupes minimales fournit par MOCUS. Il a de meilleures garanties d'approximation et doit être préféré à l'utilisation de MOCUS, lorsque l'on souhaite faire uniquement des calculs quantitatifs.
De plus, cette technique permet de faire de meilleures approximations que celles résultant du calcul de l'enveloppe monotone. Lorsque l'on calcule l'enveloppe monotone d'une fonction, on considère que les littéraux négatifs ont une probabilité d'occurrence proche de 1, c'est-à-dire que la probabilité d'occurrence des événements de base considérés est proche de 0. Or, dès que cette dernière probabilité est supérieure à 0,05, l'approximation considérée est déjà suffisamment faussée pour pouvoir être mesurée. Il est possible de localiser de tels événements de base afin de ne pas les supprimer lors de l'utilisation du quantificateur existentiel, ce qui permet de régler de manière plus fine les approximations réalisées par ce genre de technique.

6.7 BIBLIOGRAPHIE

[BBR97] M. Bouissou, F. Bruyère, and A. Rauzy. *BDD based fault-tree processing: A comparison of variable ordering heuristics*. In Proceedings of European Safety and Reliability Association Conference, ESREL'97, vol. 3, pp. 2045-2052. Pergamon, 1997

[BE65] Z.W. Birnbaum and J.P. Esary. *Modules of coherent binary systems*. SIAM J. of Applied Mathematics, vol. 13, pp. 442-462, 1965.

[BRKM91] K.M. Butler, D.E. Ross, R. Kapur, and M.R. Mercer. *Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered BDDs*. In Proceedings of the 28th Design Automation Conference, DAC'91, June 1991. San Francisco, California.

[Bry86] Bryant R. *Graph Based Algorithms for Boolean Fonction Manipulation*. IEEE Transactions on Computers, vol. 35(8), pp. 677-691, August 1986

[Bry92] Bryant R. *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*. ACM Computing Surveys, vol. 24, pp. 293-318, September.1992

[CDLR98] S. Combacon, Y. Dutuit, A. Laviron, and A. Rauzy. *Comparison between two tools (Aralia and ESCAF) applied to the Study of the Emergency Shutdown System of a Nuclear Reactor*. In Proceedings of the International Conference of Probabilistic Safety Assessment and Management, PSAM'4, vol. 2, pp. 1019-1024, New-York, 1998. Springer Verlag.

[CDRB99] E. Châtelet, Y. Dutuit, A. Rauzy, and T. Bouhoufani. *An optimized procedure to generate sums of disjoint products*. Reliability Engineering and System Safety, vol. 65, pp. 289-294, 1999.

[Cha75] P. Chatterjee. *Modularization of fault trees: A method to to reduce the cost of analysis*. Reliability and Fault Tree Analysis, SIAM, pp. 101-137, 1975.

[CY85] L. Camarinopoulos and J. Yllera. *An Improved Top-down Algorithm Combined with Modularization as Highly Efficient Method for Fault Tree Analysis*. Reliability Engineering and System Safety, vol. 11, pp. 93-108, 1985.

[DABC96] O. Dubois, P. André, Y. Boufkhad, and J. Carlier. *SAT versus UNSAT*, 1 AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 415-436, 1996.

[DR96] Y. Dutuit and A. Rauzy. *A Linear Time Algorithm to Find Modules of Fault Trees*. IEEE Transactions on Reliability, vol. 45(3), pp. 422-425, 1996.

[DR97] Y. Dutuit et A. Rauzy. *Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia*. Reliability Engineering and System Safety, vol. 58, pp. 127-144, 1997.

[DR98] Y. Dutuit and A. Rauzy. *Polynomial approximations of boolean functions by means of positive binary decision diagrams.* In Proceedings of European Safety and Reliability Association Conference, ESREL'98, 1998.

[FFK88] M. Fujita, H. Fujisawa, and N. Kawato. *Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams.* In Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'88, pp. 2-5, 1988.

[FFM93] M. Fujita, H. Fujisawa, and Y. Matsugana. *Variable Ordering Algorithm for Ordered Binary Decision Diagrams and Their Evalutation.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 12(1), pp. 6-12, January 1993.

[FMK91] M. Fujita, Y. Matsunaga, and N. Kakuda. *On the Variable Ordering of Binary Decision Diagrams for the Application of Multilevel Logic Synthesis.* In Proceedings of European Conference on Design Automation, EDAC'91, pp. 50-54, 1991.

[FS90] S.J. Friedman and K.J. Supowit. *Finding the Optimal Variable Ordering for Binary Decision Diagrams.* IEEE Transactions on Computers, vol. 39(5), pp. 710-713, May 1990.

[FYBS93] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli. *Dynamic Variable Reordering for BDD Minimization.* In Proceedings of the European Design Automation Conference EURO-DAC'93/EURO-VHDL'93, pp. 13-135, 1993.

[HK95] W. Hennings and N. Kuznetsov. *FAMOCUTN and CUTQ – Computer codes for fast analytical evaluation of large fault trees with replicated and negated gates*. IEEE Transaction on Reliability, vol. 44, pp. 368-376, 1995.

[KHI89] T. Kohda, E.J. Henley, and K. Inoue. *Finding Modules in Fault Trees.* IEEE Transactions on Reliability, vol. 38(2), pp. 165-176, June 1989.

[LGTL85] W.S. Lee, D.L. Grosh, F.A. Tillman, and C.H. Lie. *Fault tree analysis, methods and applications : a review*. IEEE Transactions on Reliability, vol. 34, pp. 194-303, 1985.

[Loc81] M.O. Locks. *Modularizing, minimizing and interpreting the K & H fault tree.* IEEE Transactions on Reliability, vol. R-30, pp. 411-415, December 1981.

[Nik00] M. Nikolskaia. *Binary Decision Diagrams and Applications to Reliability Analysis.* Thèse de doctorat de l'Université Bordeaux I (LaBRI). Janvier 2000.

[PG80] A. Pagès et M. Gondran, *Fiabilité des systèmes*, Edition Eyrolles, Collection de la Direction des Etudes et Recherches d'Electricité de France, vol. 39, ISSN 0399-4198, 1980

[Rau01b] A. Rauzy, *Mathematical Foundation of Minimal Cutsets*, LaBRI, UMR CNRS 5800, Université Bordeaux I, Technical Report LaBRI/MVTSi/Aralia/TR99-5,Submitted to IEEE Transactions on Reliability, 2001

[Rau93] Rauzy A. *New Algorithms for Fault Trees Analysis*. Reliability Engineering & System Safety, vol. 5(59), pp. 203-211, 1993

[RSS75] *Reactor Safety Study - An assessment of accident risks in US commercial nuclear power plants*. WASH 1400 (NUREG 74/014) US NRC, October 1975 (appelé aussi "Rapport Rasmussen")

[Rud93] R. Rudell. *Dynamic Variable Ordering for Ordered Binary Decision Diagrams*. In Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'93, pp. 42-47, November 1993.

[STK88] Sang Hoon Han, Tae Woon Kim, and Kun Joong Yoo. *Development of an Integrated Fault Tree Analysis Computer Code MODULE by Modularization Technique.* Reliability Engineering and System Safety, vol. 21, pp. 145-154, 1988.

[Wil85] J.M. Wilson. *Modularizing and minimizing fault trees.* IEEE Transactions on Reliability, vol. R-34, pp. 320-322, 1985.

[Yll88] J. Yllera. *Modularization methods for evaluating fault trees of complex technical systems*. In A. Kandel and E. Avni, editors, Engineering Risk and Hazard Assessment, vol. 2, chapter 5. CRC Press, 1988. ISBN 0-8493-4655-X.

7. LES APPROXIMATIONS PROBABILISTES

Ce chapitre est consacré à la dernière phase du projet Hévéa : l'étude des différentes approximations utilisées dans les logiciels de quantification des EPS.
Les chapitres précédents nous ont permis d'une part de pouvoir traduire toutes les séquences d'un arbre d'événements en formules booléennes, et d'autre part, de pouvoir quantifier de manière exacte ces dernières.
La section 7.1 recense toutes les approximations probabilistes mises en oeuvre dans les logiciels de quantification des EPS, qui, dans leur majorité, peuvent être simulées à l'aide de l'algorithmique présente dans Aralia.
La section 7.2 permet de comprendre les raisons pour lesquelles les approximations peuvent donner des résultats éloignés de ceux que l'on attend.
Pour finir, la section 7.3 est consacrée à l'étude de ces approximations appliquée à deux EPS industrielles. Ces EPS sont celles qui nous ont servi à valider la robustesse de nos approches dans les chapitres précédents. Elles sont présentées en détail au chapitre 5.5.1. Le but de cette étude est surtout de discuter des erreurs d'interprétation que peuvent entraîner ces approximations.

7.1 APPROXIMATIONS POSSIBLES

Les BDD ont l'immense avantage de pouvoir évaluer la probabilité d'une formule booléenne sans avoir besoin de calculer ses coupes minimales, contrairement aux autres algorithmes utilisés dans la plupart des logiciels disponibles sur le marché.

7.1.1 APPROXIMATION DANS L'OBTENTION DES COUPES MINIMALES

Considérons, dans un premier temps, que la formule booléenne à étudier est une fonction monotone. C'est le cas de la très grande majorité des arbres de défaillance construits manuellement, des diagrammes ou des réseaux de fiabilité. Les exceptions les plus courantes sont les séquences d'un arbre d'événements et les arbres générés automatiquement par des ateliers-logiciels tels que Cécilia WorkShop - OCAS [GH01] basé sur le langage AltaRica.
Dans le cas des formules booléennes monotones, les notions de coupe minimale et d'implicant premier sont confondues. Toute formule monotone est équivalente à la disjonction de ses coupes minimales. Nous pourrions donc calculer de manière exacte la probabilité de cette formule à partir de toutes ses coupes minimales.
Dans la pratique, dès que cette formule devient importante, les algorithmes permettant d'obtenir les coupes minimales ne s'intéressent plus qu'aux coupes minimales prépondérantes. Il y a deux manières de définir cette notion de prépondérance :

- Pour une approche qualitative, les coupes prépondérantes sont les plus courtes.
- Pour une approche quantitative, ce sont celles qui ont la plus forte probabilité.

Dans les deux cas il n'est plus possible, à partir de ces coupes minimales prépondérantes, de calculer la probabilité exacte de la formule.

L'algorithme MOCUS, utilisé dans la très grande majorité des logiciels d'évaluation des EPS, est efficace car il cherche les coupes minimales prépondérante en termes de probabilité. Notons qu'en général les probabilités associées aux pannes des composants d'un système ne sont pas égales. Aralia dispose d'un algorithme de type MOCUS, ce qui va permettre de simuler cette approximation.
L'algorithme du p-BDD_k de Aralia (Cf. chapitre 4.2) permet, quant à lui, de calculer les coupes minimales prépondérantes d'ordre inférieur ou égal à k.

7.1.2 CALCUL DE LA PROBABILITE A PARTIR DES COUPES MINIMALES

Il y a principalement deux méthodes pour calculer la probabilité d'une formule booléenne à partir de ses coupes minimales :

1. La premier méthode, généralement utilisée, consiste à appliquer le développement de Sylvester-Poincaré. Cette méthode est difficilement applicable dans son intégrité dès que le nombre des coupes minimales est important. Ce problème d'explosion combinatoire du nombre de termes à calculer oblige à ne considérer que les k premiers termes de ce développement, voire de se limiter à son premier terme. Cette simplification peut être pessimiste (si k est impair) ou optimiste (si k est pair).
2. La seconde méthode consiste à réécrire les coupes minimales en somme de produits disjoints. La probabilité est alors obtenue en additionnant les probabilités des produits disjoints.

Il est possible de calculer le premier terme du développement de Poincaré directement dans Aralia à partir d'un ZBDD. Mais il est également possible de calculer le BDD codant les implicants premiers d'un ZBDD à l'aide de l'algorithme SOP (Sum-Of-Product) de Aralia. La probabilité exacte de cet ensemble de coupes minimales peut être obtenue à partir de ce BDD.
Aralia peut donc simuler les deux méthodes de calcul de la probabilité d'une formule booléenne à partir de ses coupes minimales.

7.1.3 APPROXIMATION UTILISEE DANS LES ARBRES D'EVENEMENTS

Multiplication des probabilités rencontrées le long des séquences

Il existe encore des logiciels qui calculent, pour chaque système de sûreté, la probabilité d'occurrence de panne et qui l'injecte dans l'arbre d'événements en considérant le système de sûreté comme indépendant de tous les autres. La probabilité d'une séquence s'obtient alors simplement en multipliant les probabilités de succès ou d'échec des systèmes de sûreté sollicités au cours de cette séquence.
Notons que ces logiciels sont de plus en plus rares et sont généralement utilisés pour la qualité de leur présentation graphique des séquences accidentelles.

Une commande spécifique a été ajoutée au sein de Hévéa, afin de pouvoir effectuer les calculs avec cette approximation et ainsi d'être en mesure de l'estimer.

Ne pas considérer les branches de succès

Une seconde catégorie d'approximation consiste à ne pas tenir compte des parties négatives des séquences (correspondant aux branches de succès de l'arbre). Dans ce cas, seules les dépendances entre les systèmes de sûreté défaillants sont prises en compte.

Lorsque les formules booléennes associées aux dysfonctionnements des systèmes de sûreté sont monotones, ne pas considérer les branches de succès dans un arbre d'événements, c'est faire des approximations pessimistes à deux titres :

1. Premièrement, dans ce cas, les probabilités des parties négatives sont considérées comme étant égales à l'unité. Cette approximation se justifie si toutes les défaillances élémentaires ont une très faible probabilité d'occurrence. Or, certains événements représentés dans le cadre des EPS, comme les facteurs humains, peuvent avoir de fortes probabilités.
2. Deuxièmement, si des systèmes fonctionnent, c'est qu'il existe au minimum un chemin de succès pour ces systèmes. Un composant A ne peut pas être considéré au même instant en fonctionnement par un système S_1 et en panne pour un système S_2. Si la défaillance d'un composant A entraîne à elle seule la panne de S_1 et que pour une séquence donnée, S_1 fonctionne, c'est que le composant A fonctionne. Si ce composant est utilisé par S_2 qui lui est considéré, dans la même séquence, comme en panne, cela ne peut être imputé au composant A. En d'autres termes, en ne considérant pas les branches de succès, les coupes minimales d'une séquence, où S_1 fonctionne et où S_2 est en panne, vont faire intervenir la défaillance du composant A alors que c'est théoriquement impossible puisque A doit fonctionner pour que S_1 fonctionne.

Si les événements redoutés attachés aux branches d'échec des arbres d'événements ne sont pas relatifs à des formules monotones, il n'est pas possible de trancher sur le fait que l'approximation réalisée est optimiste ou pessimiste.

Notons que cette approximation simplifie considérablement les calculs.
Afin de simuler cette approximation, nous avons ajouté au sein de Hévéa la possibilité de ne générer que les parties positives des séquences.

Technique de *cut set matching*

L'approximation logique du précédent paragraphe pouvant difficilement se justifier, des techniques ont été mises en œuvre afin de prendre en compte les dépendances fonctionnelles entre systèmes de sûreté.
Rasmuson résume dans [Ras91] le principe permettant d'obtenir les coupes minimales d'une séquence dans les outils actuelles d'évaluation des EPS.

Une séquence d'événements est composée de *m* systèmes S_i (pour *i* de 1 à *m*) qui fonctionnent et de *n* systèmes en panne P_j (pour *j* de 1 à *n*). Les modèles logiques de défaillance des systèmes en panne sont combinés à l'aide d'une porte "Et", tandis que ceux des systèmes qui fonctionnent sont combinés à l'aide d'une porte "Ou". Soient *P* et *S* qui représentent ces deux formules booléennes.

$$P = \bigcap_{j=1}^{n} P_j \qquad et \qquad S = \bigcup_{i=1}^{m} S_i \qquad [7\text{-}1]$$

Les coupes minimales de *P* et *S* sont obtenues et calculées séparément. Les coupes minimales des systèmes défaillants sont ensuite comparées aux coupes minimales des systèmes fonctionnant. Si une coupe minimale de *P* inclut une coupe minimale de *S*, elle n'est pas à considérer et est donc supprimée de la liste des coupes minimales de P. Elle n'est pas à considérer, car si cette coupe est satisfaite, alors S l'est aussi et si S est satisfait la séquence ne peut pas avoir lieu.
Le procédé de comparaison est réalisé pour toutes les coupes minimales de P. Une fois cette opération réalisée, la liste des coupes minimales de P est de nouveau réduite.
Le procédé de comparaison des coupes minimales et de leur éventuelle suppression est appelé *cut set matching* ou *list matching*.

La notion de *Shadow variable*, proposée par Rauzy dans [Rau02a] et implémentée dans l'algorithme MOCUS de Aralia, a le même objectif que le *cut set matching*. L'idée est de marquer les *variables* représentant les branches de succès d'une séquence par un attribut *Shadow*. Lors du déroulement de l'algorithme, ces variables ne sont jamais développées. En revanche, lorsqu'une coupe minimale est déterminée, si elle satisfait une des *shadow variables* du produit en cours, c'est que cette coupe ne peut pas survenir.
L'avantage de cette implémentation est qu'il n'est pas nécessaire de calculer les coupes minimales des parties ayant fonctionné, puisque la suppression des coupes minimales, rendant "vrai" une branche de succès, est réalisée à la volée.

Facteur correctif

Un facteur correctif est utilisé afin de prendre en compte les systèmes en fonctionnement lors des calculs de probabilité. La probabilité de la séquence $p(Seq_i)$, calculée en fonction des coupes minimales précédemment obtenues, est pondérée par la probabilité que les systèmes des branches de succès S_i (pour *i* de 1 à *m*) fonctionnent. Cette dernière est égale à l'unité diminuée de la valeur de la probabilité qu'un des systèmes S_i soit en panne. Le facteur de pondération *FP* est donc donné par la formule suivante :

$$FP = 1 - p\left(\bigcup_{i=1}^{m} S_i\right) \qquad [7\text{-}2]$$

La probabilité "corrigée" est définie par $p'(Seq_i) = p(Seq_i) \times FP$
Cette probabilité "corrigée" est égale à la probabilité exacte de la séquence s'il n'y a pas de dépendances fonctionnelles entre les systèmes en panne et ceux qui fonctionnent.
Si ce n'est pas le cas, ce facteur correctif n'a plus de justification.

7.1.4 METHODES GENERALEMENT EMPLOYEES

Les meilleurs outils d'évaluation d'EPS présents sur le marché fonctionnent de la manière suivante :

- Ils déterminent les coupes minimales d'une séquence, MCS(Seq_i), en utilisant le procédé de list matching.
- Les coupes minimales des systèmes en panne, MCS($PSeq_i$), et des systèmes qui fonctionnent, MCS($SSeq_i$), sont obtenues à l'aide d'un algorithme de type MOCUS. Les coupes prépondérantes sont obtenues en ne conservant que les produits supérieurs à une valeur seuil. Ce seuil (seuil relatif) est généralement défini par rapport à la somme des probabilités des coupes minimales déjà obtenues. Une valeur comprise entre [10^{-3}, 10^{-5}] est utilisée dans la pratique afin de conserver une certaine robustesse dans l'obtention des résultats.
- La probabilité de la séquence, $p^+(Seq_i)$, est obtenue soit en appliquant le développement de Sylvester-Poincaré limité aux premiers termes sur MCS(Seq_i), soit en appliquant d'autres méthodes d'approximation.
- La probabilité associée à MCS($SSeq_i$), $p^-(Seq_i)$, est obtenue de la même manière que $p^+(Seq_i)$.
- La probabilité corrigée de la séquence, $p'(Seq_i)$ est égale à $p^+(Seq_i) \times (1 - p^-(Seq_i))$.

Tout ce processus de calcul peut être simulé à l'aide des primitives de calcul d'Aralia et de Hévéa.

7.1.5 ENVELOPPE MONOTONE

Toutes les approximations des précédents paragraphes considèrent l'enveloppe monotone de la séquence plutôt que la séquence elle-même. Cette approximation est-elle réaliste dans le cadre des arbres d'événements ? Nous avons dit plus haut que certains événements de base, comme ceux qui concernent les facteurs humains, peuvent avoir de fortes probabilités (proche de 0.1).

Pour pouvoir calculer la probabilité exacte associée à cette enveloppe monotone d'une fonction quelconque, il convient de calculer l'ensemble des coupes minimales de la formule, puis de générer le BDD codant ces coupes minimales à l'aide de l'algorithme SOP et enfin de calculer la probabilité de la séquence sur ce BDD. Cette approche requiert l'usage de trois BDD ce qui peut être coûteux en termes de mémoire à allouer.

Il existe une seconde façon de procéder, qui ne s'applique qu'aux séquences comprenant des défaillances de systèmes de sûreté décrites à l'aide de formules monotones. Ce cas est, rappelons-le, le cas le plus fréquent.
Cette seconde voie consiste à prétraiter le BDD de la formule à l'aide du quantificateur existentiel, comme cela est expliqué dans le paragraphe 6.5. Le BDD manipulé a une taille moins importante et les coupes minimales de cette formule prétraitée sont les mêmes que celles de la formule. La suite de la méthode demeure inchangée : calcul des coupes minimales, puis utilisation de SOP afin d'obtenir le BDD des coupes minimales.

Pour finir, il serait sans doute intéressant de vérifier que le calcul, utilisant le quantificateur existentiel, est une meilleure approximation que celle qui repose sur le calcul des coupes minimales.

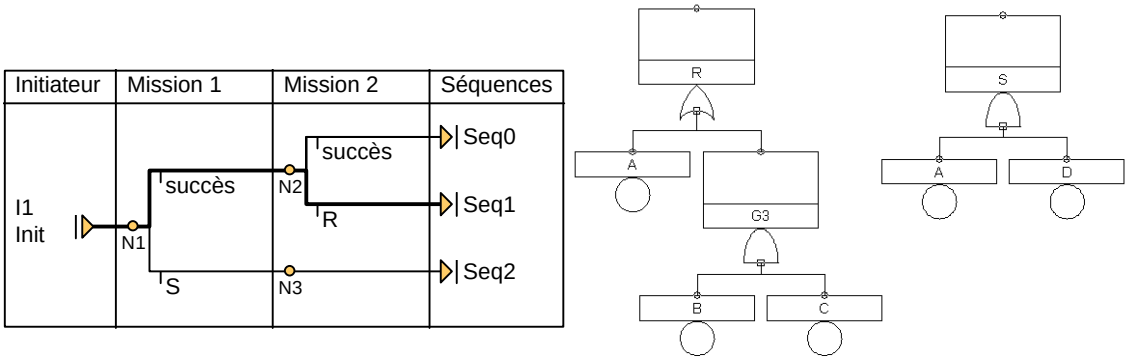
7.1.6 CONCLUSION

Les approximations au sein des arbres d'événements peuvent être nombreuses. Elles reposent toutes sur l’asymétrie entre les littéraux positifs et négatifs, c'est-à-dire qu'elles considèrent que les probabilités des littéraux positifs sont très inférieures à celles des littéraux négatifs. Cette hypothèse part du principe que les systèmes étudiés sont en général très fiables (issus par exemple de l’industrie nucléaire).
Les exemples que nous avons eu à traiter montrent cependant que cette hypothèse n'est pas toujours valide. De plus la technique, consistant à utiliser un événement de base avec un probabilité prise égale à 1 afin d'identifier les coupes qui entraînent la panne d'un sous-système donné, est utilisée par de nombreux industriels.
Dans ce cas, il est très difficile de juger des résultats obtenus, si nous ne disposons pas d'un référentiel correct. Certaines approximations réalisées par ces calculs sont pessimistes, d'autres optimistes. Comment va évoluer le résultat final ? Certains praticiens justifient ces approximations pour peu qu'elles ne perturbent pas l'importance relative des séquences. C'est l'enjeu de la troisième section de ce chapitre qui va essayer de répondre à ces questions à partir de deux EPS de taille industrielle.

7.2 ARBRES D'EVENEMENTS POSANT PROBLEME

Ce cas test a été spécialement conçu pour mettre en évidence les approximations faites habituellement lors de l'approche booléenne des arbres d'événements. Il est suffisamment petit pour que l'on puisse vérifier les calculs à la main. La structure de l'arbre et les probabilités des événements de base sont certes discutables, mais cet arbre d'événements n'a nullement pour objectif d'être représentatif des EPS faites habituellement. Son but, rappelons-le, est seulement de mettre en lumière les erreurs que peuvent engendrer les approximations de calcul.

Considérons les arbres d'événements et de défaillance ci-dessous.



Le seule séquence qui peut poser problème dans cet arbre d'événements est la séquence Seq1. Les autres ne font intervenir que des systèmes de sûreté uniquement en panne (Seq2) ou uniquement en fonctionnement (Seq0).

La formule booléenne associée à Seq1 est définie par :

$$\begin{aligned} Seq_1 &= \neg S \wedge R \\ &= \neg(A \wedge D) \wedge (A \vee (B \wedge C)) \\ &= (\neg A \vee \neg D) \wedge (A \vee (B \wedge C)) \\ &= (\neg A \wedge A) \vee (\neg A \wedge B \wedge C) \vee (\neg D \wedge A) \vee (\neg D \wedge B \wedge C) \\ &= (\neg A \wedge B \wedge C) \vee (\neg D \wedge A) \vee (\neg D \wedge B \wedge C) \end{aligned}$$

Seq1 a trois implicants premiers qui sont :

$$PI_1 = A \wedge \neg D$$
$$PI_2 = \neg A \wedge B \wedge C$$
$$PI_3 = \neg D \wedge B \wedge C$$

Notons que les coupes minimales de Seq1 sont A et BC, ce qui correspond aux coupes minimales de R.

Cet arbre d'événements a été conçu de telle manière que les techniques de *cut set matching* ne puissent pas prendre en considération les dépendances fonctionnelles entre *S* et *R*. Pour cela un événement de base fictif *D* de probabilité égale à 1 est associé à *A* ce qui rend impossible la simplification basée sur les listes des coupes minimales des succès et des échecs.
De plus, nous considérons que la probabilité des autres événements de base (*A*, *B* et *C*) est constante et égale à *q*.
A partir de ces considérations, la formule littérale de la probabilité de la séquence Seq_1 est facilement obtenue en utilisant la formule de Sylvester-Poincaré (à l'ordre 3, puisqu'il y a trois implicants premiers) :

$$\begin{aligned} p(Seq_1) &= \text{Poincaré}_1 - \text{Poincaré}_2 + \text{Poincaré}_3 \\ \text{Poincaré}_1 &= p(PI_1) + p(PI_2) + p(PI_3) \\ &= p(A)\times(1-p(D)) + (1-p(A))\times p(B)\times p(C) + (1-p(D))\times p(B)\times p(C) \\ &= (1-p(A))\times p(B)\times p(C) \qquad \text{car } (1-p(D)) = 1-1 = 0 \\ &= (1-q)\times q\times q \\ \text{Poincaré}_2 &= p(PI_1 \wedge PI_2) + p(PI_1 \wedge PI_3) + p(PI_2 \wedge PI_3) \\ &= p(A \wedge \neg D \wedge \neg A \wedge B \wedge C) + p(A \wedge \neg D \wedge \neg D \wedge B \wedge C) + p(\neg A \wedge B \wedge C \wedge \neg D \wedge B \wedge C) \\ &= p(A \wedge B \wedge C \wedge \neg D) + p(\neg A \wedge B \wedge C \wedge \neg D) \\ &= p(A)\times p(B)\times p(C)\times(1-p(D)) + (1-p(A))\times p(B)\times p(C)\times(1-p(D)) \\ &= 0 \qquad \text{car } (1-p(D)) = 1-1 = 0 \\ \text{Poincaré}_3 &= p(PI_1 \wedge PI_2 \wedge PI_3) \\ &= p(A \wedge \neg D \wedge \neg A \wedge B \wedge C \wedge \neg D \wedge B \wedge C) \\ &= 0 \\ \text{d'où} \quad p(Seq_1) &= (1-q)\times q\times q \end{aligned}$$

Cet arbre mettant en échec les techniques de *cut set matching*, la seule solution pour obtenir une probabilité exacte à partir des résultats qualitatifs est de calculer les implicants premiers associés à *Seq1*. Toute technique s'appuyant sur les coupes minimales, c'est-à-dire toutes les techniques exposées dans la section précédente aboutissent au même résultat.

$$p^{+}(Seq_1) = p(A) + p(B)\times p(C) - p(A)\times p(B)\times p(C) = q + q^2 - q^3$$

Le facteur correctif, quant à lui, est égal à 1 - la probabilité de *S*. La probabilité corrigée est donc définie par :

$$p'(Seq_1) = p^{+}(Seq_1) \times (1 - p(A)) = (q + q^2 - q^3) \times (1 - q) = q - 2q^3 + q^4$$

Le Tableau 7-1 compare $p(Seq_1)$, $p^{+}(Seq_1)$ et $p'(Seq_1)$ en fonction de différentes valeurs de q.

q	$p(Seq_1)$	$p^{+}(Seq_1)$	$p'(Seq_1)$
0.1	0.009	0.109	0.0981
0.01	9.9e-5	0.010099	0.00999801
0.001	9.99e-7	0.001000999	9.99998e-4
0.0001	9.999e-9	0.000100009999	9.9999998e-5

Tableau 7-1 : Cas-test élémentaire : comparaison de différentes approches

Contrairement aux idées reçues, l'erreur est d'autant plus grande (proportionnellement) que la probabilité des événements de base est faible. Les résultats sont heureusement pessimistes. Ils ne remettent pas en cause les EPS réalisées jusqu'à présent.
Ce simple cas test permet de comprendre pourquoi l'on peut obtenir des résultats pessimistes, voire très pessimistes lors de l'évaluation d'une EPS.

7.3 RESULTAT SUR LES DEUX EPS

Bien des sceptiques pourraient rétorquer que l'exemple précédent n'est qu'un cas d'école et que dans le cas de vraies EPS, ce phénomène est largement minimisé. Cette remarque est d'ailleurs valable pour tous les articles qui proposent des exemples d'arbres d'événements et qui essayent de justifier [Ras91] ou d'infirmer [AD00, DA00] les approximations faites à partir de ces exemples. Cela ne remet pas du tout en cause l'intérêt de ces articles, car ils permettent au moins de rappeler à la communauté le bien-fondé de cette discussion et surtout de ne pas oublier qu'il convient de rester dans le domaine d'admissibilité des méthodes que l'on emploie.
Les deux EPS étudiées aboutissent à la même conclusion. Nous avons donc allégé la lecture de cette section en transférant les résultats relatifs à la première EPS à l'annexe C.

7.3.1 METHODOLOGIE

Dans la suite de cette section, nous comparons deux probabilités, pour chaque séquence considérée. La première est la valeur de référence et la seconde la valeur de l'approximation qui est réalisée. Nous nous intéressons à l'écart séparant ces deux valeurs. La meilleure manière de représenter l'erreur pouvant être faite sur des entités pouvant avoir des ordres de grandeur différents est de l'exprimer sous la forme d'un facteur d'erreur. Nous posons donc :

$$FE = \frac{\text{Approximation étudiée}}{\text{Résultat de référence}} \qquad [7\text{-}3]$$

Si $FE > 1$, l'approximation sera pessimiste. Si *FE* est supérieur à 10, nous considérerons que l'approximation considérée pour la séquence en cours est fortement pessimiste. En effet, nous pouvons admettre des facteurs d'erreurs d'ordre 2 ou 3. Au-delà de 10, l'approximation peut engendrer des erreurs de diagnostic.
L'approximation sera optimiste lorsque le facteur d'erreur est inférieur à 1. Nous serons particulièrement attentifs aux approximations optimistes qui, elles, conduisent à ne pas considérer certaines séquences qui sont prépondérantes et qui entraînent donc des erreurs graves de correction.

Lorsque l'on désire étudier l'approximation réalisée pour un ensemble de séquences, l'une des meilleures représentations graphiques possibles reste l'histogramme sur la répartition des approximations. Différentes classes d'erreurs d'approximation ont été définies. L'idée est donc de compter le nombre de séquences dont le facteur d'erreur (*FP*) appartient à une classe considérée. Dans les cas pessimistes les classes observées sont les suivantes : [1 ; 1,001], [1,001 ; 1,01], [1,01 ; 1,1], [1,1 ; 2,5], [2,5 ; 5], [5 ; 10^{+1}], [10^{+1} ; 10^{+2}], [10^{+2} ; 10^{+3}], [10^{+3} ; 10^{+4}], [10^{+4} ; 10^{+5}], [10^{+5} ; 10^{+6}]. Les bornes des premières classes sont très proches de 1 afin de différencier les approximations légèrement pessimistes des approximations franchement pessimistes. Puis leurs

amplitudes croient de manière exponentielle afin de rendre compte de la gravité des approximations qui en résultent.

Pour certaines séquences, il est possible qu'une des deux probabilités (exacte ou approximée) n'ait pas pu être calculée, à cause d'un temps de calcul trop long ou de la limite disponible en termes de mémoire. Dans ce cas, cette séquence n'est pas retenue dans la comparaison.

Cette étude pour chaque EPS-exemple a été réalisée pour deux ensembles de séquences :

- Toutes les séquences de l'EPS.
- Les séquences dites prépondérantes, c'est-à-dire celles dont la probabilité de référence ou l'approximation considérée est supérieure à un seuil spécifié.

Les séquences prépondérantes sont donc celles qui sont systématiquement étudiées du fait de l'importance de leur probabilité.

7.3.2 MULTIPLICATION DES PROBABILITES LE LONG D'UNE SEQUENCE

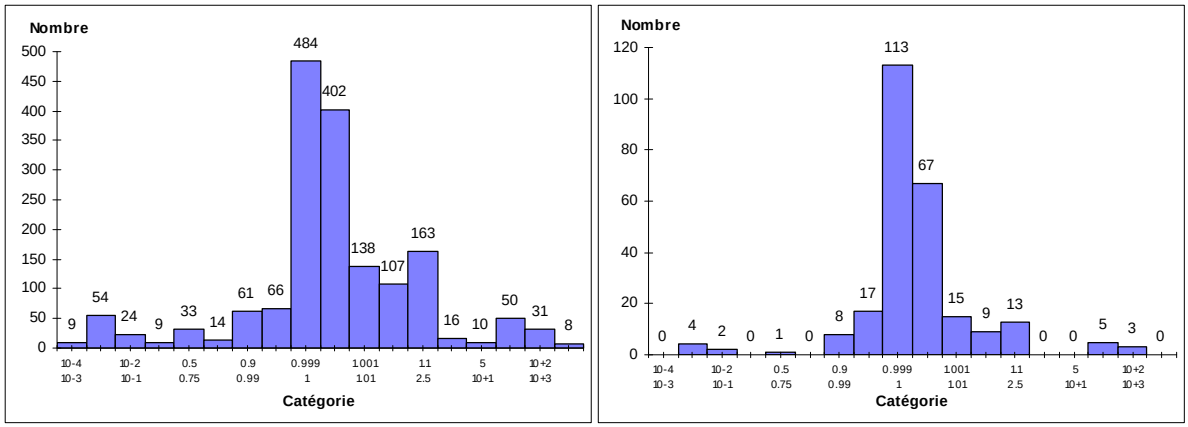


Figure 7.1 : Approximation de type "Multiplication" : Répartition du facteur d'erreur

La dispersion de l'histogramme montre bien que cette approximation ne peut pas être envisagée, lorsqu'il existe des dépendances fonctionnelles entre les systèmes de sûreté. Cela n'empêche pas de la mettre en œuvre, si les arbres d'événements sont construits à l'aide d'une méthodologie qui vise à supprimer ces dépendances fonctionnelles.

Les erreurs engendrées peuvent être pessimistes ou optimistes suivant la séquence envisagée et dans les deux cas les facteurs d'erreur maximaux sont considérables (supérieurs à 1000) et ce, même pour les séquences prépondérantes (supérieurs à 500 dans ce cas).

7.3.3 NE PAS TENIR COMPTE DES SYSTEMES DE SURETE QUI FONCTIONNENT

Les résultats ont été obtenus en comparant la probabilité exacte de la séquence en mettant ou non "à vrai" les systèmes de sûreté ayant fonctionné.

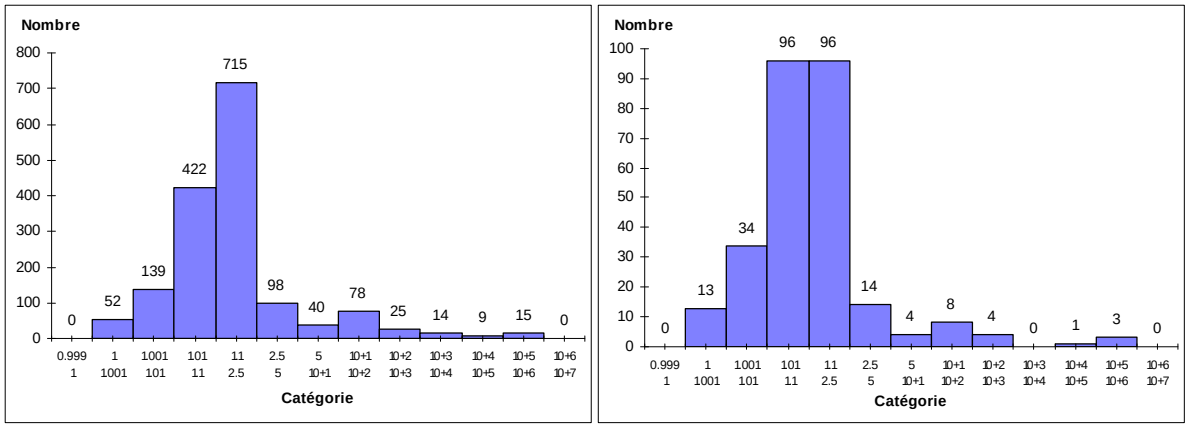


Figure 7.2 : Approximation de type "Exist-True" : Répartition du facteur d'erreur

Les résultats obtenus sont pessimistes, puisqu'ils ne prennent pas en compte la non-défaillance de certains événements de base. Les résultats présentés sur la Figure 7.2 montrent que l'évaluation des séquences, même prépondérantes, peut être très pessimiste (facteur d'erreur supérieur à 10^{+5}). En clair, une séquence peut avoir sa probabilité évaluée à 10^{-7} alors que sa valeur exacte est de l'ordre de 10^{-12}. Cette approximation n'est donc pas dangereuse en soit, puisque qu'elle ne donne jamais de résultats optimiste. Elle est juste irréaliste, car pour deux séquences de même ordre de grandeur, elle ne permet pas de prédire laquelle des deux est la moins probable.

7.3.4 ENVELOPPE MONOTONE

La probabilité exacte de l'enveloppe monotone de chaque séquence a été obtenue en utilisant trois BDD successifs comme indiqué au paragraphe 7.1.5. On calcule donc le facteur d'erreur de chaque séquence à partir de :

$$FE = \frac{\Pr\left(\mathrm{SOP}\left(\mathrm{MCS}\left(\mathrm{BDD}(Seq)\right)\right), t\right)}{\Pr\left(\mathrm{BDD}(Seq), t\right)} \qquad [7\text{-}4]$$

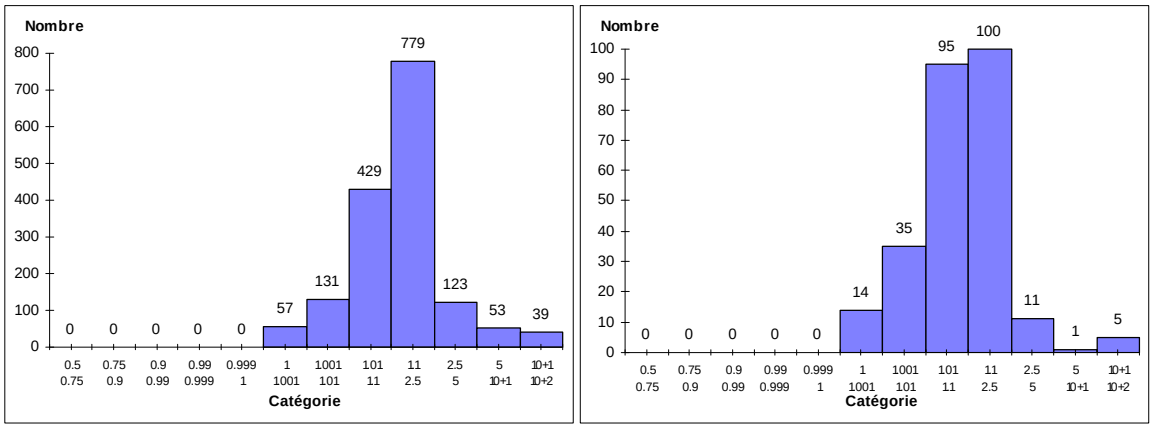


Figure 7.3 : Approximation "Enveloppe monotone" : Répartition du facteur d'erreur

Les résultats confirment, d'une certaine manière, le phénomène qui a été mis en évidence dans le cas-test de la section 7.2. Dans cette section, nous avions montré qu'il était possible de construire un arbre d'événements fictif qui, lorsque l'on ne considère que son enveloppe monotone, fournit une évaluation probabiliste pessimiste, voire très pessimiste suivant la valeur des lois des probabilités des événements de base. Nous en avions conclu qu'il était possible de trouver de tels phénomènes dans les arbres d'événements industriels.
Sur l'EPS2, il y a ainsi 5 séquences prépondérantes qui ont un facteur d'erreur supérieur à 10. Le plus grand facteur d'erreur est égal à 100, lorsque l'on considère toutes les séquences de l'étude, et égal à 25, si l'on ne considère que les séquences prépondérantes. Toutes les méthodes de calcul se basant sur l'enveloppe monotone d'une fonction booléenne entraîneront au minimum cette approximation pessimiste.

D'ailleurs, les calculs sur ces séquences à l'aide d'un quantificateur existentiel, entraînent des résultats très proches car ils sont basés sur la même idée : supprimer les événements de base non survenus.

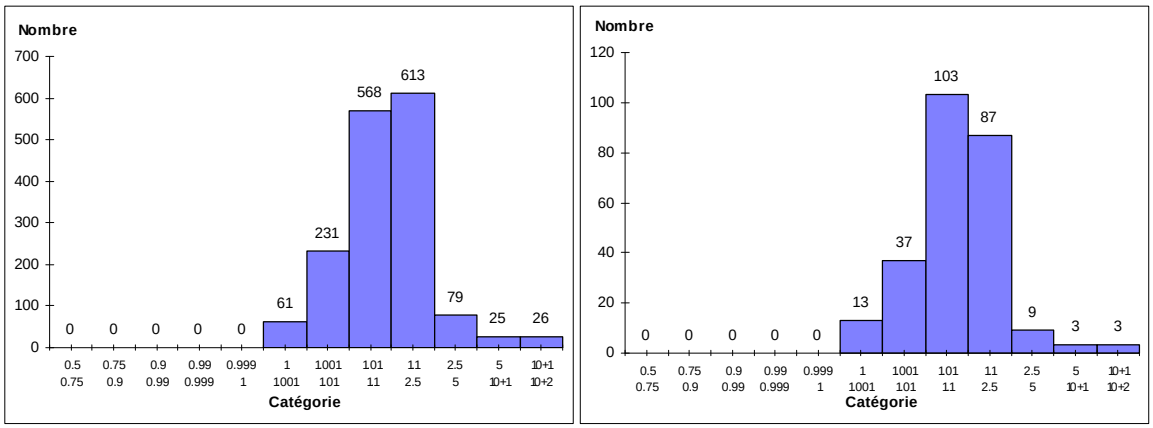


Figure 7.4 : Approximation "Quantificateur existentiel" : Répartition du facteur d'erreur

Le facteur d'erreur étudié est alors donné par :

$$FE = \frac{\Pr\left(\text{BDD}\left[\text{exist}\left(\text{leaves}\left(Seq^{-}\right) \setminus \text{leaves}\left(Seq+\right); Seq^{-}\right) \wedge Seq^{+}\right], t\right)}{\Pr\left(\text{BDD}\left(Seq\right), t\right)} \qquad [7\text{-}5]$$

Avec exist(E, f) : quantificateur existentiel de f par rapport à l'ensemble des variables appartenant à E.
leaves(f) : opérateur retournant l'ensemble des feuilles (variables terminales) de la formule f.
$E_1 \setminus E_2$: différence ensembliste.
Seq^- : Disjonction des systèmes de sûreté en fonctionnement de la séquence Seq.
Seq^+ : Disjonction des systèmes de sûreté en panne de la séquence Seq.

7.3.5 APPROXIMATION DE LA PROBABILITE

Nous souhaitions étudier les erreurs de calcul lorsque l'on évalue la probabilité d'une fonction booléenne en sommant les probabilités de toutes ses coupes minimales. Seule l'approximation concernant cette évaluation nous intéresse. En conséquence, le facteur d'erreur étudié s'obtient à partir de l'expression suivante :

$$FE = \frac{\text{PP}\left(\text{MCS}\left(\text{BDD}\left(Seq\right)\right), t\right)}{\Pr\left(\text{SOP}\left(\text{MCS}\left(\text{BDD}\left(Seq\right)\right)\right), t\right)} \qquad [7\text{-}6]$$

Avec PP(E) : Probabilité dite "de Poincaré" calculé sur un ensemble de coupes minimales (Cf. chap.4.2).

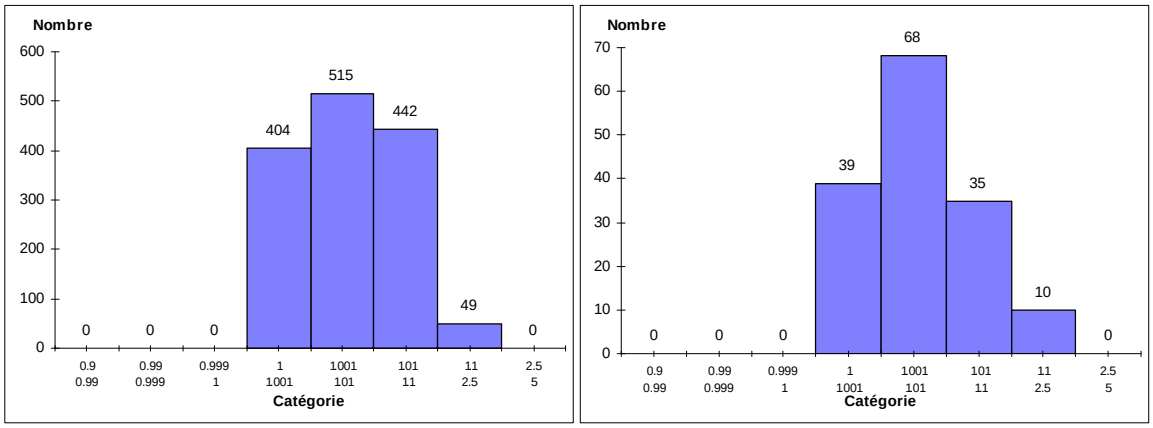


Figure 7.5 : Approximation "Somme de la probabilité des produits" : Répartition du facteur d'erreur

Le premier terme du développement de Sylvester-Poincaré donne des résultats pessimistes qui sont très proches de la valeur de référence. Le facteur d'erreur maximum est de 2,06 (la moyenne étant de 1,03) ce qui, au vu des grandeurs manipulées, n'est pas significatif.

7.3.6 OBTENTION DES COUPES MINIMALES

Algorithme de type "MOCUS"

Encore une fois, nous avons souhaité n'étudier que les approximations pouvant résulter de l'obtention des coupes minimales à l'aide de l'algorithme de type "MOCUS". Nous avons donc voulu faire ce calcul sur un ensemble de formules monotones. De cette manière, les coupes minimales d'une formule considérée sont également ses implicants premiers et l'évaluation probabiliste sur ses coupes minimales n'est pas perturbée par une approximation sur l'enveloppe monotone de la fonction considérée. Rappelons que les algorithmes de type MOCUS ne peuvent déterminer que les coupes minimales d'une fonction et pas ses implicants premiers.
Le facteur d'erreur est alors donné par :

$$FE = \frac{\Pr\left(\text{SOP}\left(\text{MOCUS}\left(Seq^{+}, seuil\right)\right), t\right)}{\Pr\left(\text{BDD}\left(Seq^{+}\right), t\right)} \qquad [7\text{-}7]$$

Avec Seq^+ : Disjonction des systèmes de sûreté en panne de la séquence *Seq*.
seuil : Seuil relatif utilisé par l'algorithme MOCUS afin de ne récupérer que les séquences prépondérantes.

Plusieurs valeurs de *seuil* ont été testées : 10^{-3}, 10^{-4}, 10^{-5}. Pour chacune de ces valeurs, les résultats sont très similaires, mais d'autant meilleurs que le seuil est faible. Les résultats fournis par l'algorithme de type MOCUS de ce chapitre correspondent à un seuil relatif de 10^{-4}.

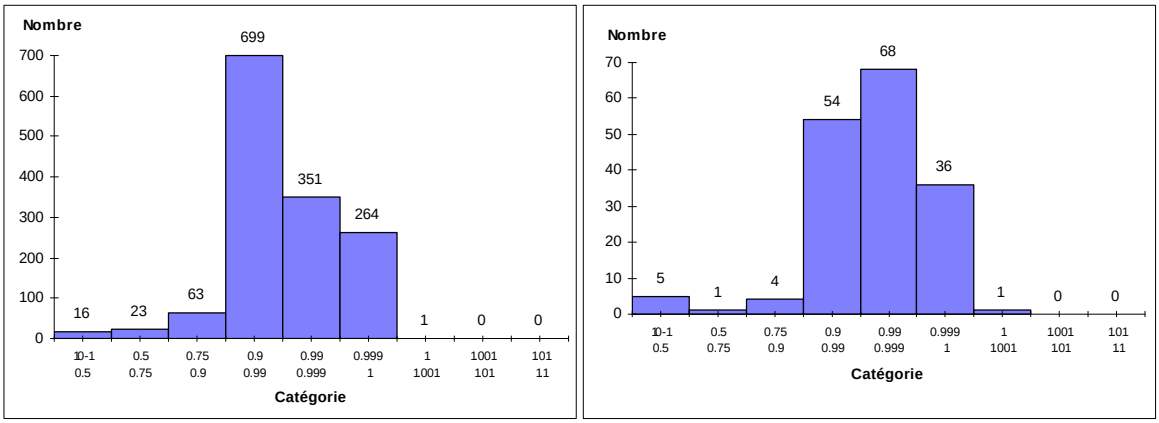


Figure 7.6 : Approximation "Mocus sur des fonctions monotones" : Répartition du facteur d'erreur

Comme nous pouvions nous y attendre, ces résultats sont optimistes. En effet, l'algorithme MOCUS ne conserve que les coupes minimales prépondérantes, il ignore donc les coupes minimales non prépondérantes.
Le facteur d'erreur minimal est égal 0,484, ce qui correspond à un facteur légèrement supérieur à 2 entre la probabilité évaluée et la probabilité exacte.

Principe du *Cut Set Matching* ou des *Shadow Variables*

Le paragraphe 7.1.3 présente ces deux notions et montre qu'elles sont globalement équivalentes. Dans Aralia, seule la notion des *Shadow variables* a été implémentée.
Le facteur d'erreur considéré est défini par :

$$FE = \frac{\Pr\left(\mathrm{SOP}\left(\mathrm{MOCUS}^*\left(Seq, seuil\right)\right), t\right)}{\Pr\left(\mathrm{BDD}\left(Seq\right), t\right)} \qquad [7\text{-}8]$$

Avec MOCUS* : Algorithme de type "MOCUS" utilisant le principe des *Shadow variables* sur les systèmes de sûreté de la séquence ayant fonctionné
seuil : Seuil relatif utilisé par l'algorithme MOCUS, afin de ne récupérer que les séquences prépondérantes

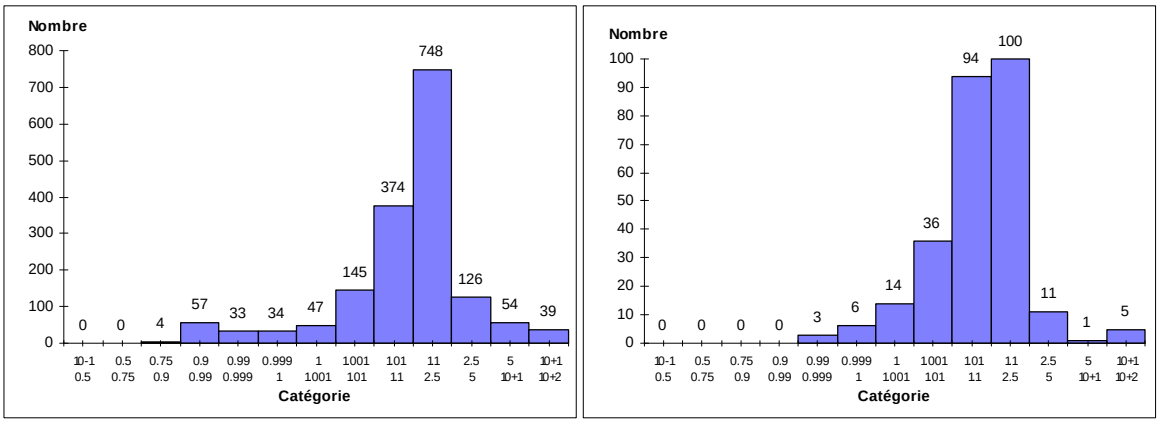


Figure 7.7 : Approximation "Mocus avec Shadow Variable" : Répartition du facteur d'erreur

Les résultats considérés sont globalement pessimistes. Ils sont d'ailleurs très proches de ceux obtenus sur le calcul de l'enveloppe monotone ce qui est, somme toute, assez logique.
En fait, ces résultats montrent que les algorithmes de type "MOCUS" utilisant la notion de *cut set matching* engendrent deux approximations : ils considèrent l'enveloppe monotone de la formule (approximation pessimiste) et ne considèrent que ces coupes prépondérante (approximation optimiste). L'expérimentation montre que la

première approximation est plus importante que la seconde. Bien que certaines évaluations de séquences soient légèrement optimistes, la majorité est pessimiste. Le facteur d'erreur moyen est d'autant plus faible que le seuil relatif de l'algorithme MOCUS l'est.
Le facteur d'erreur pessimiste maximum est le même que celui associé aux approximations relatives à l'enveloppe monotone. Les erreurs de diagnostic peuvent donc être engendrées de la même manière.

7.3.7 APPROXIMATION "OPERATIONELLE" (GENERALEMENT EMPLOYEE)

Le facteur d'erreur considéré pour cette expérimentation est défini par :

$$FE = \frac{\mathrm{PP}\left(\mathrm{MOCUS}^{*}\left(Seq, seuil\right), t\right)}{\Pr\left(\mathrm{BDD}\left(Seq\right), t\right)} \qquad [7\text{-}9]$$

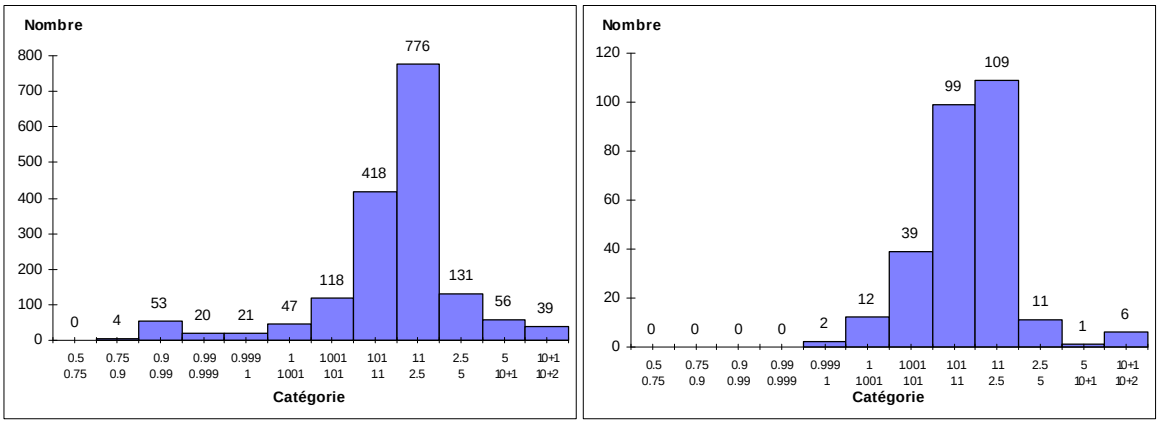


Figure 7.8 : Approximation "Opérationnelle" : Répartition du facteur d'erreur

Les résultats (Cf. Figure 7.8) de l'approximation opérationnelle (somme des probabilités des coupes minimales obtenues à partir d'une technique *list matching* résultant d'un algorithme MOCUS) sont très proches de ceux de la Figure 7.7. Ce qui n'a rien d'étonnant compte tenu des résultats des approximations sur le calcul de la probabilité du paragraphe 7.3.5.

En revanche, les résultats utilisant le facteur correctif sont suffisamment irréalistes pour ne pas considérer cette approximation dans la pratique. Si tous les résultats précédents paraissent conformes à nos attentes, cette approximation est difficilement justifiable, tant dans sa définition que dans ses résultats.
Le facteur d'erreur considéré est défini par :

$$FE = \frac{\Pr\left(\mathrm{BDD}\left(Seq\right), t\right)}{\mathrm{PP}\left(\mathrm{MOCUS}^{*}\left(Seq, seuil\right), t\right) \times \left[1 - \mathrm{PP}\left(\mathrm{MOCUS}\left(Seq^{-}, seuil\right), t\right)\right]} \qquad [7\text{-}10]$$

où Seq^{-} représente la conjonction des systèmes de sûreté fonctionnant de la séquence *Seq*.

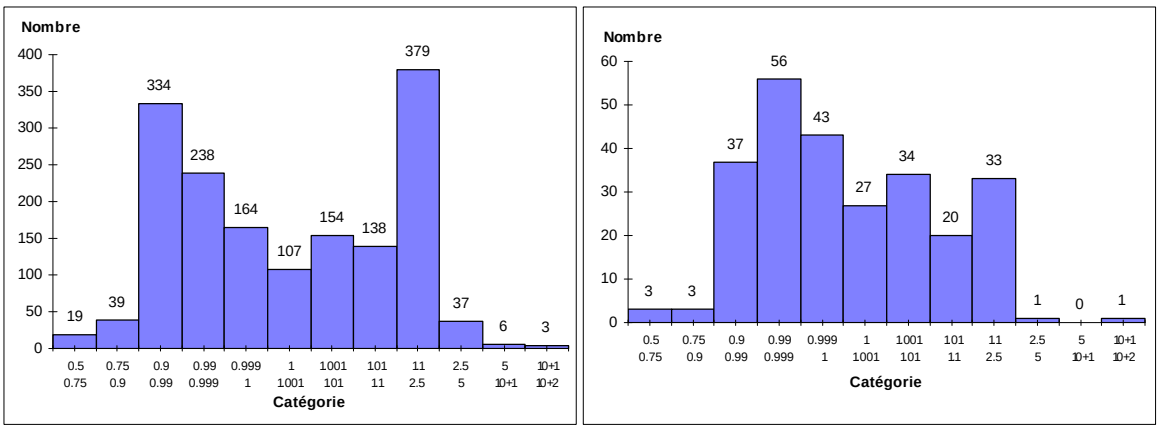


Figure 7.9 : Approximation "Opérationnelle avec facteur correctif" : Répartition du facteur d'erreur

7.4 CONCLUSION

Ce chapitre a permis d'analyser les différents types d'approximations généralement utilisés dans les logiciels de quantification des EPS. Les approximations historiques, comme la multiplication des probabilités des systèmes de sûreté rencontrés le long d'une séquence, ne peuvent pas être utilisées lorsqu'il existe des dépendances fonctionnelles entre les systèmes de sûreté. Les approximations engendrées par l'utilisation de l'algorithme MOCUS, ou par l'évaluation de la probabilité obtenue en additionnant les probabilités des coupes minimales, n'ont pas été mises en défaut sur les expérimentations réalisées dans le cadre de cette étude. En réalité, la seule approximation posant réellement un problème est due au fait que l'algorithme MOCUS (ou les autres techniques considérées) renvoie les coupes minimales de la formule considérée et non les implicants premiers. En ne considérant que l'enveloppe monotone de la fonction booléenne, les résultats sont pessimistes, voire très pessimistes. La section 7.2 a montré, à partir d'un cas-test, de quelle manière de tels résultats peuvent être obtenus.

L'étude sur les deux EPS de référence permet donc deux conclusions :

- Les approximations des logiciels usuellement utilisés sont pessimistes. Le facteur d'erreur maximum est égal à 25 si on ne considère que les séquences ayant une probabilité évaluée supérieure à 10^{-7} (il est égal à 100 si le seuil considéré est de 10^{-8}).
- Les approximations ne sont pas conservées d'une séquence à l'autre. C'est-à-dire que pour deux séquences quelconques de l'arbre, les facteurs d'erreur ne sont pas équivalents.

La résultante de ces deux points est qu'il est possible de faire des erreurs de diagnostic non négligeables sur les EPS étudiées. Pour illustrer ce propos, considérons un organisme de tutelle qui fixe comme règle que toutes les séquences ayant une probabilité de 10^{-7} et aboutissant à des conséquences jugées inacceptables doivent être réévaluées et éventuellement améliorées. Au vu des probabilités évaluées, ces organismes de tutelle vont émettre des demandes de modification en réalité injustifiées sur des séquences qui paraissent ne pas répondre aux critères imposés. Ces approximations engendrent donc des erreurs de diagnostic qui peuvent avoir un coût non négligeable.

De plus, les approximations ne permettent pas de garantir une hiérarchisation des séquences en fonction de leurs probabilités. Même si cette hiérarchisation est globalement respectée, certaines séquences ne sont pas de même rang suivant que l'on considère ou non l'enveloppe monotone de la séquence. On peut observer dans l'annexe D que la $113^{ème}$ séquence prépondérante, lorsqu'on prend en compte l'approximation de la probabilité, n'est en réalité que la $180^{ème}$ si on calcule la probabilité exacte.

Pierre-Etienne Labeau souligne de plus que :"Si le résultat de référence pris en compte ici intègre bien les dépendances fonctionnelles, les dépendances temporelles restent peu considérées. Or, celles-ci sont susceptibles de modifier à nouveau l'ordre de grandeur de la probabilité de certaines séquences, voire d'ajouter des séquences supplémentaires."

7.5 BIBLIOGRAPHIE

[AD00] J. Andrews and S. Dunnett. *Event-Tree Analysis Using Binary Decision Diagrams*. IEEE Transactions on Reliability, vol. 49, N°2, pp. 230-238, June 2000

[DA00] S. Dunnett & J. D. Andrews. *Improving Accuracy in Event Tree Analysis*, in Proceedings of ESREL 2000, SARS and SRA-EUROPE annual Conference, pp. 15-17, May 2000

[GH01] J. Gauthier, T. Hutinet. Atelier Cécilia - Outil de Conception et d'Analyse Système (O.C.A.S.) Manuel utilisateur. *Dassault - IXI* Version 2.5, 2002.

[Ras92] D. M. Rasmuson. *A comparison of the small and large event tree approaches used in PRAs*. Reliability Engineering and System Safety, vol. 37 pp. 79-90, 1992

Partie III :
Projet Aloès

Le but d'un outil ou d'une méthode d'allocation de fiabilité est d'aider à la conception ou à l'amélioration de systèmes de plus en plus complexes dans un environnement économique de plus en plus concurrentiel. L'allocation de fiabilité vise à trouver un juste équilibre entre différentes grandeurs caractéristiques d'un système donné, tel que son coût ou sa fiabilité. Il existe fondamentalement deux approches d'allocation.
La première est une démarche descendante. Elle consiste à attribuer des objectifs à chaque composant d'un système, en fonction de son type et de paramètres de pondération, de telle façon que l'objectif global, au niveau du système, soit atteint.
La seconde, plus générale, est une démarche ascendante. Elle consiste à chercher une solution répondant à des critères d'optimalité en jouant sur des variables de décision, qui représentent les choix d'organisation, de qualité des composants, de politique de maintenance, d'architecture système..., que l'équipe de conception souhaite étudier. Les paramètres fiabilistes des composants, leurs coûts élémentaires,... sont directement définis en fonction de ces variables de décision.
Nous nous sommes principalement intéressés à la seconde approche dite de l'allocation par optimisation.
Les méthodes d'optimisation proposées dans la littérature sont nombreuses et présentent toutes des singularités. Elles se différencient sur de nombreux points tels que :

- le type de systèmes pris en compte (série - parallèle, quelconque,...),
- les modèles fiabilistes considérés (arbres de défaillance, graphes de Markov,…),
- la (ou les) grandeur(s) à optimiser,
- la prise en compte ou la non prise en compte de contraintes (de poids, de coût, de fiabilité,…),
- la nature des variables de décision (discrètes ou continues),
- le type de fonction permettant de définir les choix,
- les algorithmes d'optimisation, ...

La difficulté, pour une équipe de conception, est de trouver, parmi toutes les méthodes d'optimisation disponibles, celle qui sera plus adaptée au type de problème qu'elle a à résoudre.

Ne souhaitant pas imposer aux utilisateurs une formalisation particulière du problème d'allocation, nous avons proposé de le décrire à l'aide d'un langage : le langage Aloès. La seule limitation que nous avons retenue concerne le type de modèles fiabilistes utilisés. Ce sont les modèles booléens d'analyse des risques, c'est-à-dire les arbres de défaillance et les réseaux de fiabilité.
Ce travail s'inscrit dans le cadre d'un partenariat entre différents organismes (le LaBRI d'une part, ALSTOM, CEA-IPSN, EDF, Elf-EP, Technicatome et IXI d'autre part). Ce partenariat s'est fixé comme objectif la réalisation d'un logiciel d'optimisation associé aux logiciels de traitement des formules booléennes Aralia et de traitement des réseaux de fiabilité Réséda.
Le logiciel Aloès est un interprèteur de commandes prenant en entrée des descriptions de problèmes d'allocation écrites dans le langage Aloès. Il offre trois algorithmes de résolution qui se différencient par leur caractère exhaustif ou stochastique ainsi que par la nature des variables de décision qu'ils peuvent prendre en considération (variables discrètes ou continues).
Le langage Aloès et les algorithmes considérés sont présentés au chapitre 10. Ils ont été validés sur de nombreux exemples de la littérature. Le chapitre 11 est consacré à une étude d'allocation sur un ensemble de réseaux de fiabilité. Cette étude aborde trois problèmes différents : l'allocation de redondance, l'optimisation de maintenance et l'allocation en conception préliminaire.
En marge de cette étude, nous nous sommes intéressés à un aspect plutôt méthodologique qui concerne la notion de coût d'un système et de sa définition en fonction du coût des composants. Le chapitre 9 présente une étude des différentes fonctions de coût proposées dans la littérature.

8. PRESENTATION

Lors de la conception de systèmes industriels, les autorités de certification, les donneurs d'ordres fixent des objectifs en termes de fiabilité, de performance, de coût, de poids, de volume … Les sociétés chargées de réaliser ces systèmes doivent donc faire des choix entre différentes architectures, différents composants, différentes technologies … de façon à optimiser la fiabilité, les performances, le coût, le poids ou le volume.

Le terme "Allocation d'exigence" regroupe toutes les méthodes utilisées pour faire ces choix de façon optimale, ou semi-optimale. On trouve aussi dans la littérature le terme "Allocation de fiabilité" parce que la grandeur à optimiser est souvent la fiabilité du système.

Une des premières à avoir été mise en œuvre est la méthode d'égale allocation. Elle permet de bien comprendre le principe de base d'une allocation de fiabilité. Elle s'applique à un système série de n composants. A partir d'un objectif de fiabilité R_{Obj}, elle consiste à allouer la même fiabilité à tous les composants du système. La fiabilité allouée au composant i est donnée par la relation suivante :

$$R_i = \left(R_{Obj}\right)^{\frac{1}{n}} \qquad i = 1..n \qquad [8\text{-}1]$$

La fiabilité du système est égale à l'objectif qui était fixé.

$$R_S = \prod_{i=1}^{n} R_i = \prod_{i=1}^{n} \left(\left(R_{Obj}\right)^{\frac{1}{n}} \right) = R_{Obj} \qquad [8\text{-}2]$$

Cette méthode simpliste est évidemment très discutable. Elle ne s'applique qu'à des systèmes série, ce qui est un peu limitatif, d'autant qu'elle ne prend aucunement en considération les particularités de chaque composant du système (état de l'art, degré de complexité, sollicitation, environnement ...).

Par la suite, les problèmes d'allocation ont suscité un large intérêt de la part de la communauté scientifique, d'où une littérature très abondante sur ce sujet. De nombreux états de l'art existent dans ce domaine [Dhi86, MLR92, AC94, EA98, PK00]. Ils essayent principalement de faire une classification entre les différentes méthodes proposées. Cette classification se fait en fonction de différents critères, tels que la structure du système (série, série/parallèle, formule booléenne ...), la formalisation du problème, la méthode de résolution, le domaine d'application ...
Ces articles distinguent deux familles d'allocation de fiabilité : les méthodes de pondération et les méthodes d'optimisation. Dans les sections qui vont suivre, nous nous efforcerons de présenter brièvement les principes généraux de ces deux familles.

8.1 ALLOCATION DE TYPE PONDERATION

Les articles [AC94] et [EA98] sont des états de l'art ne s'intéressant qu'aux allocations de pondération et à leur mise en œuvre. Cloarec et Allain-Morin affirment, dans leur conclusion, que les méthodes d'optimisation ne sont pas à ranger dans la catégorie des méthodes d'allocation, car, dans ces dernières, le critère à optimiser n'est en général pas un critère fiabiliste. Nous verrons dans le chapitre 8.2 que l'objectif de ces méthodes est tout de même d'améliorer les systèmes en prenant en considération des grandeurs fiabilistes.
La remarque de ces auteurs montre que ces deux types d'allocation bien, qu'ayant la même finalité, n'ont pas beaucoup de points communs.

8.1.1 PRINCIPE GENERAL

D'une manière synthétique, les méthodes de pondération :

- s'appuient sur une modélisation du système par arbre de défaillance ou diagramme de fiabilité, voire sur une simple équation décrivant la grandeur fiabiliste considérée;
- cherchent à atteindre un objectif pour cette grandeur fiabiliste;
- répartissent l'effort en termes de fiabilité sur les différents composants du système.

Elles restent donc dans un cadre purement fiabiliste.
La grandeur fiabiliste considérée peut être la fiabilité ou la disponibilité du système, son taux de défaillance ou même son temps moyen de réparation [AR80].
La répartition de l'effort se fait en fonction de paramètres de pondération associés aux composants. Ces paramètres sont généralement issus de caractéristiques techniques [HMS85], telles que la complexité ou la maturation du sous-système, ses conditions environnementales, son aptitude à être réparé, ...
Ces méthodes ont généralement l'avantage d'être simples à mettre en œuvre, tant du point de vue informatique, que du point de vue méthodologique.

8.1.2 QUELQUES EXEMPLES

La méthode ARINC

Développée en 1956 aux Etats-Unis par Aeronautical Radio INCorporation, cette méthode a un certain nombre de limites. Elle ne s'applique qu'à des systèmes série. Les temps de mission de tous les sous-systèmes doivent être égaux. L'objectif à atteindre est donné sous la forme d'un taux de défaillance.
La méthode alloue un taux de défaillance à chaque sous-système de la manière suivante :
Soit l_i le taux de défaillance du sous-système i estimé à l'aide d'un retour d'expérience ou d'un jugement d'expert.
On assigne à chaque sous-système une grandeur v_i proportionnelle à l_i, calculée à partir de :

$$v_i = \frac{l_i}{\sum_{j=1}^{n} l_j} \qquad i = 1,...,n \qquad [8\text{-}3]$$

Les taux de défaillance alloués sont définis par :

$$l_i^* = v_i \times l_{obj} \qquad i = 1,...,n \qquad [8\text{-}4]$$

Cette méthode conserve la relation d'ordre pour la fiabilité des sous-systèmes. Cette méthode est particulièrement adaptée à l'expertise d'un système existant.

La méthode d'allocation de BRACHA

La méthode BRACHA est adaptée aux systèmes série. Les facteurs de pondération utilisés lors de l'allocation de l'objectif de fiabilité sont au nombre de quatre pour chaque sous-système : son niveau de connaissance (a_{Ai}), sa complexité (a_{Xi}), ses conditions environnementales (a_{Ei}) et son temps de mission (a_{Ti}).
La fiabilité allouée pour chaque sous-système est donnée par :

$$R_i^* = R_{obj}^{W_i} \qquad i = 1,...,n \qquad [8\text{-}5]$$

avec

$$W_i = \frac{d_i}{\sum_{j=1}^{n} d_j} \qquad i = 1,...,n \qquad [8\text{-}6]$$

et

$$d_i = a_{Ai} \times \left(a_{Xi} + a_{Ei} + a_{Ti} \right) \qquad i = 1,...,n \qquad [8\text{-}7]$$

Les facteurs de pondération sont eux-mêmes définis par des paramètres plus techniques. Ainsi le facteur a_{Ti} est défini comme étant le rapport du temps de mission du système sur le temps de mission du sous-système i.
Comme le souligne [AC94], le nombre de caractéristiques pris en compte n'est pas forcément limité. C'est le principe de la méthode qui est intéressant, puisqu'il permet de considérer les réalités techniques du système.

La méthode d'Arsenault-Roberts [AR80]

Cette méthode est une des rares méthodes d'allocation s'intéressant à la maintenance des systèmes. Son but est de répartir un temps moyen de réparation (MTTR) au niveau système sur ses différents sous-systèmes. Elle ne s'applique qu'à des systèmes série dont tous les composants sont réparables. Son principe est très proche de la méthode BRACHA, dans le sens où elle se base sur des caractéristiques techniques liées à la maintenance. La référence [BDL88] présente cette méthode de manière assez complète.
Le temps moyen de réparation alloué pour chaque sous-système est donné par :

$$MTTR_i^* = \frac{K_i}{K_S} \times MTTR_{Obj} \qquad i = 1,...,n \qquad [8\text{-}8]$$

avec $$K_S = \frac{\sum_{i=1}^{n} l_i \times K_i}{\sum_{i=1}^{n} l_i} \qquad [8\text{-}9]$$

et $$K_i = \frac{\sum_{j=1}^{p} K_i^j}{p} \qquad i = 1,...,n \qquad [8\text{-}10]$$

K_i est donc la somme arithmétique de p facteurs secondaires définis par l'utilisateur. Ces facteurs reflètent l'aptitude du sous-système à être réparé : Ils peuvent correspondre à la "détectabilité", l'accessibilité, la "manutentionabilité", la complexité, la "démontenabilité" des sous-systèmes. Ils sont définis par jugement d'expert en donnant une note comprise entre 1 et 10. Une matrice d'analyse est donc constituée pour l'ensemble des sous-systèmes.

L'Allocation Par Pondération (APP) de ARPO

L'allocation par pondération proposée dans [BB97a, BB97b] cherche à atteindre un objectif global d'indisponibilité pour un système. Cette indisponibilité est calculée à l'aide d'un arbre de défaillance. Cette allocation recherche un compromis entre une allocation se basant exclusivement sur un retour d'expérience et une allocation qui ne tiendrait compte que de la structure du système.
En d'autres termes, on associe à chaque composant C_i, un facteur d'allocation noté K_i défini par

$$K_i = \frac{q_i}{C + \mathrm{DIF}(ER, E_i)} \qquad i = 1,...,n \qquad [8\text{-}11]$$

Avec :

- q_i l'indisponibilité du composant i, déterminée à partir du retour d'expérience ou par jugement d'expert,
- DIF(ER, E_i) = p(Ei | ER), le facteur d'importance de diagnostic calculé à partir de l'arbre de défaillance décrivant l'événement redouté ER du système et des probabilités q_i associées aux défaillances des composants de base (E_i),
- C une constante supérieure à zéro permettant de trouver un compromis entre le retour d'expérience pris en compte par q_i et la structure du système prise en compte par le facteur d'importance du composant.

On déduit les indisponibilités à allouer aux différents composants en résolvant les équations suivantes à l'aide d'une méthode itérative :

$$q_i^* = m \times K_i \qquad i = 1,...,n \qquad [8\text{-}12]$$

$$Q_S(q_1^*, q_2^*, \ldots, q_n^*) \approx Q^* \qquad [8\text{-}13]$$

Avec :

- Q^*, l'objectif global d'indisponibilité,
- Q_S : l'indisponibilité du système calculée à partir de l'arbre de défaillance décrivant l'événement redouté du système et des probabilités q_i^* associées aux défaillances des composants de base,
- m une constante multiplicative.

Les auteurs de cette méthode précisent qu'il est nécessaire de tester plusieurs valeurs pour C afin de choisir celle qui permet le meilleur compromis.

8.1.3 GENERALISATION

Les trois premières méthodes présentées dans le paragraphe précédant ne fonctionnent que pour des systèmes série. Se limiter à cette architecture de systèmes restreint fortement le domaine d'utilisation de ces méthodes. En revanche faire une pondération en fonction de caractéristiques techniques très générales pouvant être définies par jugement d'expert est une approche intéressante qu'il ne faut pas négliger. L'allocation par pondération présentée par Bouissou et Bourgade s'applique à tout système dont l'événement redouté peut être défini à l'aide d'un arbre de défaillance. Cependant, seuls un paramètre de retour d'expérience et la structure du système sont pris en compte. La complexité d'amélioration des composants, comme les autres caractéristiques techniques, n'est pas considérée.

Ces méthodes peuvent être formalisées de la manière suivante.
Soit G la grandeur à pondérer fonction d'un certain nombre de paramètres modifiables p_i (i de 1 à n) et d'une structure non modifiable S. Chaque paramètre p_i est défini par une expression $Fct_i(m)$ prenant ou non en compte un paramètre de pondération m. L'allocation par pondération cherche m tel que

$$p_i = Fct_i(m) \qquad i = 1,...,n$$

$$G(S, p_1, ... p_n) = Obj + \varepsilon \qquad [8\text{-}14]$$

Où Obj est la valeur de G à pondérer et ε est une valeur tendant vers 0. La pondération peut être réalisée par résolution itérative à l'aide d'une dichotomie sur m. L'expression $Fct_i(m)$ est définie en fonction du système et des paramètres que l'on souhaite prendre en compte.
Ce type de mise en œuvre permet de ne pas faire de restriction sur la méthode de pondération à employer. Chacun peut ainsi définir sa propre méthode de pondération : retour d'expérience, jugement d'experts ...

L'exemple suivant propose une amélioration de la fonction de l'APP de ARPO, car elle permet de spécifier des bornes de probabilité pour chaque événement de base.
Soit ER un événement redouté décrit à l'aide d'un arbre de défaillance composé de n événements de base (e_1, ..., e_n). La probabilité de ER, grandeur à pondérer, est fonction de l'arbre de défaillance et de l'indisponibilité p_i des événements de base. L'expression $Fct_i(m)$ décrivant l'indisponibilité de l'événement e_i est définie à l'aide de l'équation suivante :

$$Fct_i(m) = \text{Borne}\left(\frac{m \times q_i}{C + \text{DIF}(ER, e_i)}, min_i, max_i \right) \qquad [8\text{-}15]$$

avec
- m : paramètre de pondération
- q_i : Indisponibilité de e_i par retour d'expérience
- min_i : Indisponibilité minimum de e_i
- max_i : Indisponibilité maximum de e_i
- C : constante fixée par des experts
- Borne : fonction qui borne la valeur de la première expression en fonction des 2 suivantes
- DIF(ER,e) : Facteur d'importance de diagnostic de l'événement e sur la variable sommet ER.

Remarque : En posant pour un événement de base j, $min_j = max_j = q_j$, on force l'algorithme à prendre comme indisponibilité associée à l'événement i, la valeur de son retour d'expérience. L'événement i n'est alors plus pondéré.

8.2 ALLOCATION DE TYPE OPTIMISATION

Les méthodes d'optimisation considèrent d'une manière ou d'une autre n variables dites de décision. Ces dernières sont des variables bornées discrètes ou continues. Elles correspondent, par exemple, à la probabilité de défaillance ou au taux de défaillance des équipements composant le système étudié pour les variables

continues, ou encore au nombre de composants à prendre en compte dans une redondance active pour les variables discrètes. Des grandeurs physiques sont liées à ces variables de décision. Il peut s'agir de la fiabilité ou la disponibilité du système, de son "coût" ... . D'autres grandeurs sont quelquefois prises en considération en tant que contraintes du système comme le volume, le poids ou la consommation de ressources.
Le processus d'optimisation consiste alors à trouver les valeurs des variables de décision qui minimisent ou maximisent une des grandeurs physiques du système, tout en respectant une ou plusieurs contraintes sur les autres grandeurs physiques. En général, il s'agit de minimiser le coût de l'installation sous contrainte de fiabilité (ou disponibilité) ou de l'exercice dual, c'est-à-dire maximiser la fiabilité du système sous contrainte de coût.
Ces méthodes d'optimisation correspondent à de l'allocation de fiabilité, lorsqu'au moins une des grandeurs (à optimiser ou à contraindre) est une grandeur fiabiliste.

8.2.1 MODELES FIABILISTES UTILISES

Comme il s'agit d'allocation de fiabilité, les modèles permettant le calcul de grandeurs fiabilistes sont ceux de la sûreté de fonctionnement. Ces grandeurs fiabilistes doivent être calculées de manière rapide - les processus d'optimisation demandant un grand nombre d'itérations - et de manière exacte ou au minimum reproductible.
Historiquement, les méthodes ne s'appliquaient que sur des modèles fiabilistes simples, comme les systèmes série, sur lesquels il est aisé de calculer les grandeurs fiabilistes en fonction des paramètres de ces équipements. Dans les année 1980, des méthodes de résolutions exactes et approchées ont vu le jour afin de traiter les systèmes de type série/parallèle, parallèle/série, k-parmi-n/série. Ces systèmes sont toujours étudiés de nos jours, car ils ne nécessitent pas de logiciel de quantification, puisque le calcul de la fiabilité repose sur une formule littérale.
Depuis que les diagrammes binaires de décision sont utilisés en fiabilité et permettent des calculs quantitatifs rapides et exacts, des modèles booléens quelconques peuvent maintenant être étudiés.
Nous pouvons noter aussi quelques utilisations des graphes de Markov [ACNT99] permettant de prendre en considération de manière plus réaliste les politiques de maintenance et les redondances passives.
Enfin pour finir, quelques auteurs proposent pour le calcul de grandeurs fiabilistes une simulation de Monte Carlo [CMZ00]. La simulation pose des problèmes de temps de calcul et, parfois, de précision. En revanche, cette approche permet de prendre en compte des systèmes ayant de fortes dépendances tant temporelles que fonctionnelles.

8.2.2 FORMALISATION DU PROBLEME

La formalisation du problème d'allocation par optimisation proposée en début de ce chapitre est très générale et correspond à ce que vise à long terme le projet Aloès. Les formalisations habituelles du problème d'optimisation sont plus restrictives.
Dans ce paragraphe, nous considérons que le problème est d'améliorer la fiabilité du système sous contrainte de coût. Il est facile, d'un point de vue formalisation, de considérer le problème dual et/ou d'ajouter d'autres contraintes. Dans la suite de ce paragraphe, on considère un système *S* composé de *n* équipements.

Les premières méthodes d'optimisation proposaient d'améliorer la fiabilité de *S* en améliorant la fiabilité des équipements. A chaque équipement est associé une variable de décision x_i continue, variant entre 0 et 1, représentant la probabilité de défaillance du composant.

Le problème devient alors :

Maximiser [P 8-1]

$$R_S = f_R(x_1,\ldots,x_n)$$

Sous les contraintes

$$Cost(x_1,\ldots,x_n) \le Cost_{ref}$$

$$0 \le x_j \le 1, \qquad j = 1,\ldots,n$$

où $f_R(x_1,\ldots,x_n)$ est une fonction donnant la probabilité que le système fonctionne.

$Cost(x_1,\ldots,x_n)$ est une fonction donnant le coût de l'installation.

En général le coût de l'installation est défini comme la somme des coûts des équipements. Le coût d'un équipement est défini par une fonction dépendant de sa fiabilité. Afin de simplifier les calculs, beaucoup d'auteurs considèrent que la fonction de coût des composants est la même pour tous. Cette considération n'est pas satisfaisante, car chaque équipement a un coût qui croît différemment en fonction de sa fiabilité.
Au lieu de considérer une variation continue de la fiabilité des équipements, une autre approche consiste à faire pour chaque équipement un choix entre un nombre fini (et en général petit) de solutions potentielles. Une variable de décision discrète ch_i est associée à chaque équipement. Elle correspond aux différents termes de l'alternative pour l'équipement considéré.

Une autre manière d'améliorer la fiabilité de S est d'ajouter des niveaux de redondance pour chaque équipement. Chaque équipement est alors composé de t_i composants mis en redondance active. t_i est une variable de décision discrète variant entre min_i (min_i >= 1) et max_i, représentant le nombre de redondances souhaité pour l'équipement *i*. Le coût et la fiabilité d'un composant de base pour l'équipement i sont deux constantes notées respectivement c_i et r_i. Les composants de base sont supposés non réparables.
Le problème devient alors :

Maximiser P 8-2

$$R_S = f_R(t_1,\ldots,t_n)$$

Sous les contraintes

$$Cost(t_1,\ldots,t_n) \le Cost_{ref}$$

$$\min_j \le t_j \le \max_j, \qquad j = 1,\ldots,n$$

Le coût de l'installation est défini comme étant la somme des coûts des équipements :

$$Cost(t_1,\ldots,t_n) = \sum_{i=1}^{n} (t_i \times c_i) \qquad [8\text{-}16]$$

En général, la fiabilité d'un équipement (noté R_i) est donnée par la relation :

$$R_i = 1 - \prod_{j=1}^{t_i} (1 - r_i) \qquad [8\text{-}17]$$

L'équation 8-17 ne prend pas en considération les défaillances de mode commun qui sont pourtant loin d'être négligeables pour des composants identiques mis en redondance.
[Lib96] rapporte le cas de composants identiques mis en redondance d'ordre 1, 2 et 3 pour lesquels le retour d'expérience a donné les résultats suivants :

Nombre de composants	Défiabilité observée	Défiabilité théorique
1	10^{-2}	10^{-2}
2	3×10^{-4}	10^{-4}
3	10^{-4}	10^{-6}

Tableau 8-1 : Prise en compte des défaillances de cause commune

Au vu de ses résultats, on peut comprendre qu'il est peu judicieux de négliger les défaillances de cause commune, même si la plupart des méthodes proposées le font.

Pour conclure, précisons qu'une grande majorité des méthodes d'optimisation partent d'une des deux formalisations mentionnées en début de paragraphe ou d'une combinaison de celles-ci.

8.2.3 METHODES DE RESOLUTION

Il serait illusoire de vouloir traiter de toutes les méthodes de résolution de ces problèmes. Un récent état de l'art [KP00] recense plus de 80 méthodes pour résoudre uniquement les problèmes d'allocation de redondance.
Nous allons faire dans ce chapitre un rapide tour d'horizon des différentes méthodes utilisées pour résoudre les problèmes d'optimisation.

Heuristiques

La plupart des heuristiques développées pour résoudre le problème P8-2 ont un fonctionnement commun : à chaque itération, une solution est obtenue à partir de l'itération précédente en modifiant la valeur d'une seule variable de décision en fonction de facteurs de sensibilité. Dans le cas de l'allocation de redondance, il s'agit d'incrémenter le nombre de redondances pour un système donné.
Par exemple, les auteurs de [YH85] cherchent à maximiser la fiabilité d'un système sous contrainte de coût. Pour cela le facteur de sensibilité permettant de choisir le prochain composant à améliorer est égal au rapport entre le gain possible de fiabilité et la perte en terme de coût. De cette manière, les composants qui améliorent le plus le système en dégradant le moins son coût sont prioritaires. Cette heuristique, bien que non optimale, propose des solutions de bonne qualité, proches de l'optimum.

Métaheuristiques

Depuis quelques années, les métaheuristiques ont été appliquées avec succès aux problèmes d'allocation de fiabilité. Ces métaheuristiques, développées en recherche opérationnelle pour résoudre certaines instances de problèmes NP-Complets, incluent les algorithmes génétiques, le recuit simulé, la recherche tabou ...
Les algorithmes génétiques [CS98, RD98, Ele00, CMZ00] essayent d'imiter le phénomène biologique de l'évolution des espèces en sélectionnant les individus d'une génération à l'autre suivant des règles visant à favoriser les "meilleurs" individus.
Le recuit simulé [EYAC99] est basé sur un procédé physique de métallurgie.
Les métaheuristiques ont l'avantage d'être des algorithmes robustes qui n'ont besoin que de très peu d'informations pour pouvoir être mises en œuvre.

Méthodes exactes

Les méthodes d'optimisation exactes cherchent à obtenir l'optimum global d'un problème. Les algorithmes présentés dans les paragraphes précédents proposent des optima sans certitude sur l'exactitude de leur solution.
Les méthodes exactes sont basées sur des techniques de programmation mathématique comme le Branch & Bound [NNH78, SC99], l'énumération implicite [MS91, PK00] ou la programmation dynamique.
Notons que ces techniques ne résolvent que des problèmes d'allocation de type discret comme l'allocation de redondance.

8.3 AUTRES TYPES D'ALLOCATION

Nous avons regroupé dans ce chapitre d'autres problèmes d'allocation qu'il nous semble intéressant de mentionner.

8.3.1 SYSTEMES AVEC MODE DE DEFAILLANCE COMPETITIF

Jusqu'à présent, la mise en redondance d'équipement nous permettait *a priori* d'améliorer la fiabilité de notre système. Cependant, il existe des systèmes où la redondance d'équipements peut, dans certain cas, entraîner une diminution des performances du système.
Par exemple un réseau composé de n relais, mis en parallèle et habituellement fermés, a la propriété de ne pas s'ouvrir à la sollicitation si l'un des relais reste fermé suite à une défaillance "bloqué en position". En revanche, il faut que les n relais aient tous une défaillance d'ouverture intempestive pour que le réseau soit ouvert, alors qu'il ne devrait pas l'être.

Dans un réseau série-parallèle, l'ouverture intempestive de tous les relais d'un seul sous-système ouvre le système et le blocage en position fermée d'un relais de chacun des sous-systèmes empêche l'ouverture du système.
En fonction du taux de défaillance associé à chaque mode de défaillance de ces composants, la redondance peut améliorer ou diminuer la fiabilité globale du système.
Si un coût est associé à chaque défaillance du système (ouverture intempestive, bloqué fermé) en fonction de ses conséquences, le problème d'optimisation cherche à minimiser le coût de fonctionnement du système.
Les problèmes d'allocation de ce type sont traités dans la littérature [PP91, Pha92] comme des problèmes à part. Cela est dû au fait que les algorithmes permettant de les traiter (d'une manière exacte ou à l'aide d'heuristiques) sont spécifiques.

8.3.2 ORDONNANCEMENT DE COMPOSANTS INTERCHANGEABLES

Considérons un système ayant des composants interchangeables. Cela a un sens si les composants sont multifonctionnels, ou lorsqu'il existe des composants similaires ayant des données fiabilistes (taux de défaillance, indisponibilité, ...) différentes. Dans ce cas, la fiabilité du composant dépend de sa fonction, ou plus simplement, de sa position dans le système. La fiabilité du système dépend alors de l'affectation des composants à des positions requises.
Considérons un système cohérent ayant n composants interchangeables 1, 2, ..., n. On note $f(a_1, ..., a_n)$ la fiabilité du système lorsque le composant de la position i a une probabilité a_i. La fiabilité d'un composant peut dépendre de sa position au sein du système. La fiabilité du composant j lorsqu'il est affecté à la position i est noté $p_{i,j}$. Si le composant v_i est affecté à la position i pour i de 1 à n, alors la fiabilité du système est donnée par $f(p_{1,v_1}, ..., p_{n,v_n})$. Le problème est de trouver une affectation des composants $(v_1, ..., v_n)$ qui maximise $f(p_{1,v_1}, ..., p_{n,v_n})$.
Ce problème a été étudié par de nombreux auteurs [PR98] sur des systèmes série/parallèle, parallèle/série ou à partir de diagrammes de fiabilité. La majorité des solutions proposées sont des heuristiques. La plupart considèrent que la fiabilité d'un composant ne dépend pas de sa position : $p_{i,j} = r_j$ pour tous (i , j).

8.4 CONCLUSION

Ce chapitre a permis de faire un rapide tour d'horizon des méthodes et algorithmes d'allocation d'exigence en sûreté de fonctionnement. L'objectif de ces méthodes est d'aider les équipes de conception par la recherche de compromis entre les exigences du cahier des charges et les différentes options possibles pour répondre à ce cahier des charges.
Les deux principales familles d'allocation de fiabilité sont décrites succinctement ci-après:

1. Les méthodes par pondération : Elles sont à objectif fiabiliste unique. Les données permettant de définir les différents choix technologique sont "facilement" accessibles (retour d'expérience, jugement d'expert sur des caractéristiques techniques, ...). La pondération, à proprement parlé, peut être réalisée par résolution itérative (comme une dichotomie, par exemple).
2. Les méthodes par optimisation : Elles peuvent être à objectifs multiples. En général, un des objectifs correspond à la minimisation du coût du système. Le chapitre suivant montrera la difficulté d'appréhension de cette notion de coût. Le recueil des données associées aux choix envisagés est l'une des difficultés inhérentes à ces méthodes. Les algorithmes devant être mis en oeuvre pour résoudre les problèmes d'optimisation posés par ces méthodes sont loin d'être triviaux et dépendent fortement du type de problèmes posés.

8.5 BIBLIOGRAPHIE

[AC94] G. Allain-Morin et J.M. Cloarec. *L'allocation d'objectifs de sûreté de fonctionnement en Phase de Spécification et de Conception.* Actes du congrès lm9-ESREL94, pp43-52, 1994

[ACNT99] S. Allibe, A. Cabarbaye, L. Ngom et L. Tomasini. *Apports des algorithmes génétiques à la Sûreté de Fonctionnement et à l'optimisation des systèmes.* 3ème Congrès internationnal Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita 99, 1999, pp 67-77.

[AR80] J. Arsenault, J. Roberts. *Reliability & Maintainability of Electronic Systems*. Computer Science Press. 1980

[BB97a] M. Bouissou and E. Bourgade. *Evaluation et allocation d'indisponibilité à la conception des tranches nucléaires : Méthodes et Outils.* Revue Française de Mécanique n°1997-2, 1997, pp105-111

[BB97b] M. Bouissou and E. Bourgade. *Unavailability Evaluation and Allocation at the Design Stage for Electric Power Plant : Methods and Tools.* Proceeding Annual Reliability and Maintenability Symposium IEEE, 1997, pp91-99

[BDL88] Baudot, Dhoury et Leray. *Méthodologie d'allocation de fiabilité et de maintenabilité d'un système complexe*. Actes du 6^{ième} colloque national de Fiabilité et Maintenabilité, λμ6, 1988, pp. 759-766.

[CMZ00] M. Cantoni, M. Marseguerra and E. Zio. *Genetic algorithms and Monte Carlo simulation for optimal plant design.* Reliability Engineering and System Safety 68, 2000, pp29-38

[CS98] D. Coit and A. Smith. *Redundancy Allocation to Maximize a Lower Percentile of the System Time-To-Failure Distribution.* IEEE Transactions on Reliability Vol. 47 No. 1, March 1998, pp79-87

[Dhi86] B. Dhillon. *Reliability Apportionment/Allocation : A survey.* Microelectron. Reliab., Vol. 26 No. 6, 1986, pp1121-1129, 1986

[EA98] C. Elegbede and K.H. Adjallah. *State of the art in reliability allocation : A comparative study on a mechanical system*. Safety and Reliability, Lydersen, Hansen & Sandtorv (eds), 1998 Balkema, Rotterdam, pp501-508

[Ele00] C. Elegbede. *Contribution aux méthodes d'allocation d'exigences de fiabilité aux composants de systèmes.* PhD thesis, Université de Technologie de Compiègne, 2000

[EYAC99] C. Elegbede, F. Yalaoui, K. Adjallah et C. Chu. *Allocation de fiabilité par les Algorithmes Génétiques et le Recuit Simulé*. Actes du 3ème Congrès internationnal Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita 99, 1999, pp 475-483.

[HMS85] M.J. Haire, J.G. Maltese, and R.G. Sohmer. *A System Availability Top-Down Apportionment Method*. In Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'85, pp. 306-314, 1985.

[KP00] W. Kuo and V. Prasad. *An Annoted Overview of System-Reliability Optimization*. IEEE Transactions on Reliability Vol. 49 No. 2, June 2000, pp176-187

[Lib96] J. Libmann. *Eléments de sûreté nucléaire.* Les éditions de la physique, ISBN : 2-86883-274-1, 1996

[MLR92] A.A Mohamed, L. Leemis and A. Ravindran. *Optimization techniques for system reliability : a review*. Reliability Engineering and System Safety 35, 1992, pp137-146

[MS91] K. Misra and U. Sharma. *An efficient Algorithm To Solve Integer-Programming Problems Arising In System-Reliability Design*. IEEE Transactions on Reliability Vol. 40 No. 1, April 1991, pp81-91

[NNH78] Y. Nakagawa, K. Nakashima and Y. Hattori. *Optimal Relibability Allocation by Branch-and-Bound Technique*. IEEE Transactions on Reliability Vol. R-27 No. 1, 1978, pp31-38

[Pha92] H. Pham. *Optimal Design of Parallel-Series Systems with Competing Failure Modes*. IEEE Transactions on Reliability Vol. 41 No. 4, December 1992, pp583-587

[PK00] V. Prasad and W. Kuo. *Reliability Optimization of Coherent Systems.* IEEE Transactions on Reliability Vol. 49 No. 3, September 2000, pp323-330

[PP91] H. Pham and M. Pham. *Optimal Designs of {k, n-k+1}-out-of-n:F Systems (Subject to 2 Failures Modes).* IEEE Transactions on Reliability Vol. 40 No. 5, December 1991, pp559-562

[PR98] V. Prasad and M. Raghavachari. *Optimal Allocation of Interchangeable Components in a Series-Parallel System*. IEEE Transactions on Reliability Vol. 47 No. 3, September 1998, pp255-260

[RD98] Y. Ren and J. Dugan. *Design of Reliable Systems Using Static & Dynamic Fault Trees*. IEEE Transactions on Reliability Vol. 47 No. 3, September 1998, pp234-244, 1998

[SC99] C. Sung and Y. Cho. *Branch-and-Bound Redundancy Optimization for a Series System with Multiple-Choice Constraints.* IEEE Transactions on Reliability Vol. 48 No. 2, June 1999, pp108-117

[YH85] K. Yasutake-Baker and E. Henley. *A Heuristic Method to Upgrade System Availability for Hot and Cold Standby and Voting Systems*. IEEE Transactions on Reliability Vol. R-34 No. 4, 1985, pp389-392

9. LA NOTION DE COUT

Une des difficultés majeures de l'allocation d'exigences est de trouver une relation entre les contraintes ou grandeurs à optimiser et les variables de décision du problème.
Cette relation est facilement identifiable pour les grandeurs fiabilistes, grâce notamment aux modèles de la sûreté de fonctionnement. Il en est tout autrement pour les autres grandeurs considérées et plus particulièrement pour le coût de l'installation.
Par exemple, si l'on suppose qu'il existe une relation entre les variables de décision du problème et le poids (en réalité masse en kg) de chaque composant du système, il est alors aisé de définir le poids du système comme étant la somme des poids de ces composants. Pouvons-nous alors utiliser la même approche pour définir son coût ? De plus, qu'entend-on par coût d'un système ?
Après quelques réflexions sur la notion de coût d'un système, nous présenterons un certain nombre de fonctions de coût citées dans la littérature.

9.1 REFLEXIONS SUR LA NOTION DE COUT D'UN SYSTEME

Si l'on s'intéresse uniquement aux coûts d'acquisition, on va prendre en compte les coûts d'achat des composants, et peut-être ceux d'installation, mais on va négliger fortement les coûts de maintenance associés à ces composants ainsi que les coûts des consommables, c'est-à-dire les coûts de fonctionnement. Contrairement au coût d'acquisition, les coûts de fonctionnement dépendent du temps. Ils sont donc à prendre en considération sur une période donnée (coût annuel de fonctionnement). Le fonctionnement d'une installation coûte effectivement de l'argent, mais il permet aussi d'en gagner. Qui dit non fonctionnement d'une installation, dit perte de production.
D'autant plus qu'il faut également prendre en considération les coûts de démantèlement de l'installation qui, suivant la nature du système, ne peuvent pas être négligés. EDF, par exemple, a prévu un budget pour financer le démantèlement des centrales nucléaires françaises. Or, le programme de démantèlement a une durée supérieure à un siècle et il existe de nombreuses incertitudes sur l'estimation de son coût. La durée du programme pose le problème de sa pérennité. Si on peut espérer que dans 120 ans le niveau de connaissances techniques et scientifiques permettra de réduire le coût final du démantèlement, il est beaucoup plus difficile d'estimer le contexte politique, économique et social qui pourra soit accélérer le processus, soit au contraire le stopper.

Ainsi, le seul coût qui a une véritable signification est le coût du cycle de vie, c'est-à-dire le coût d'une installation de sa conception à son démantèlement ; ce qui inclut les coûts d'installation, de fonctionnement et d'indisponibilité. Les entreprises ont des difficultés à déterminer le coût global de possession d'une installation (LCC : Life Cycle Cost). Il serait utopique de chercher à optimiser dans sa globalité le coût du cycle de vie d'une installation pour deux principales raisons. La première est relative à la définition du modèle : les variables de décision représentant les choix stratégiques dans la vie d'un système seraient trop nombreuses pour pouvoir être prises en compte dans le modèle. La seconde est relative à la difficulté de traitement qui en résulterait. En effet, le modèle généré serait d'une telle taille et d'une telle complexité, du fait du nombre élevé de variables de décision concernées, que les temps de calcul et l'espace mémoire à allouer ne permettraient pas de lancer un processus d'optimisation.

Le coût d'une installation (ou d'un système) dépend donc de nombreux paramètres :

- conception,
- achat et installation,
- entretien (maintenance),
- fonctionnement (consommable),
- indisponibilité (perte de production),
- encombrement (location des locaux),
- ...

Ces différents paramètres montrent qu'il est difficile de définir le coût d'une installation simplement à partir des coûts d'équipements et qu'en conséquence de nombreux autres paramètres sont à prendre en compte.
Par ailleurs, il peut exister des contraintes globales à respecter. Dans des grosses installation, il est nécessaire d'acheter les composants par lots pour des raisons économiques. Des composants identiques pouvant entraîner des défaillances de cause commune, ils devront être utilisés sur des systèmes différents (et pas en redondance pour un même système). Une telle contrainte peut obliger à construire un modèle servant de support à l'allocation qui fasse intervenir tous les systèmes de l'installation, bien qu'ils soient indépendants du point de vue des défaillances.

Les problèmes d'optimisation supposent presque toujours que le coût du système est simplement défini comme la somme des coûts de ses composants. Le coût de chaque composant est alors défini en fonction des variables de décision qui lui sont attachées. Cette hypothèse est raisonnable lorsque l'on ne considère que le coût d'acquisition du système. Dans ce cas, les prix dégressifs des composants achetés par lots ne sont pas pris en compte.
Par contre, cette vision est trop simpliste pour prendre en compte facilement d'autres phénomènes, comme le coût d'une politique de maintenance, qu'il faut comparer au gain de productivité qu'elle permet.

9.2 FONCTIONS DE COUT CONTINUES

De nombreux articles sur l'allocation de fiabilité proposent ou recensent [Ele00] des fonctions de coût continues. Mais aucun auteur ne s'est risqué à les étudier dans un cadre opérationnel. Un des objectifs du projet Aloès étant la réalisation de guides méthodologiques, nous avons souhaité étudier la pertinence de la plupart des fonctions de coût de la littérature.
Ces fonctions définissent le coût d'un composant en fonction de sa fiabilité en supposant que ce composant n'a qu'un mode de défaillance. Les variables de décision du problème sont alors directement les fiabilités des composants. Le coût de l'installation est la somme des coûts des composants considérés.

Le principal argument en faveur de ces fonctions de coût est qu'en pratique il est difficile de définir le coût d'un composant. Elles donnent donc une forme générale et paramétrique du coût et peuvent être vues comme une première approximation de ce dernier.

Le paragraphe 9.2.1 présente quelques propriétés fondamentales des fonctions de coût. Une fonction de coût élémentaire est présentée au paragraphe 9.2.2. Le paragraphe 9.2.3 recense les différentes fonctions de coût de la littérature. Pour finir, le paragraphe 9.2.4 est consacré à une fonction de coût "empirique".
On désignera par R la fiabilité du composant et par a et b les paramètres de la fonction de coût, ces derniers étant fixés par l'utilisateur.

9.2.1 PROPRIETES REQUISES

Une fonction de coût définit le coût d'un composant en fonction de sa fiabilité.
Les modèles de coût proposés reposent sur trois propriétés fondamentales :

- La fonction de coût du composant doit être une quantité strictement positive.
- La fonction de coût doit être croissante. *A priori*, plus un composant est fiable, plus il est cher.
- La fonction de coût doit croître rapidement au voisinage de 1. Un composant ne tombant jamais en panne a un coût extrême.

A contrario, un composant souvent en panne doit être bon marché.

Dans le meilleur des cas, l'utilisateur dispose d'un certain nombre de points (couples "fiabilité - coût") provenant du retour d'expérience, de la littérature ou de composants similaires et il souhaite que la fonction de coût du composant passe par ces points de référence. Une méthode de type "moindres carrés" doit permettre de définir correctement les paramètres de la fonction de coût afin qu'elle passe par ces points.
Un des points pris en compte peut être le *seuil de fiabilité* d'un composant au delà duquel le composant ne peut techniquement pas être construit. Il suffit d'associer un coût maximal au seuil de fiabilité pour disposer

d'un point limite de la fonction de coût. Ce seuil peut aussi être fixé en ajoutant une contrainte au problème d'allocation. Par ailleurs, une fonction de coût donnée fixe intrinsèquement son propre seuil de fiabilité.
Il est important de remarquer que ce n'est pas si facile d'obtenir des points de référence. Lors des phases de conception, les ingénieurs en fiabilité disposent en général d'un seul point de référence et quelquefois d'une limite de faisabilité technique pour chacun de leur composant ou sous-système. Ils utilisent donc, à partir de ces maigres données, des fonctions permettant une première approximation.

9.2.2 FONCTION ELEMENTAIRE

La fonction de coût la plus élémentaire est définie de la façon suivante :

$$C_{El}(R) = \frac{a}{1-R} \qquad [9\text{-}1]$$

Dans la majorité des fonctions de coût proposées, le paramètre *a* est le paramètre d'échelle de la fonction. Ce paramètre, strictement positif, agit comme un coefficient multiplicateur. Il permet à la fonction de coût de passer par un point significatif.

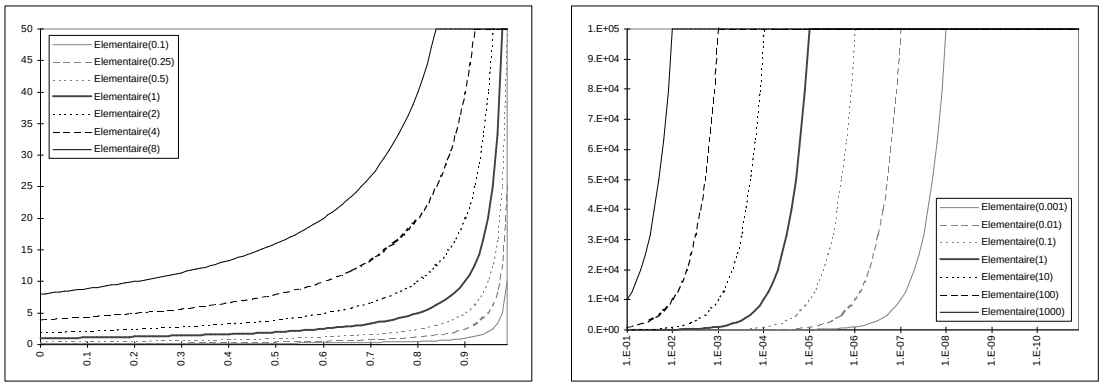


Figure 9.1 : Fonctions de coût élémentaires (a = 0.1 ; 0.5 ; 1 ; 2 ; 3 ; 4)

Bien que le domaine de définition de la fiabilité soit une grandeur comprise entre 0 et 1, les composants auront, il faut l'espérer, une fiabilité plutôt comprise entre 0,8 et 1. La forme de la fonction sur l'intervalle [0 ; 0,8] n'a donc que peu de conséquences sur le résultat d'une allocation. La défiabilité (Q=1-R) est plus facile à manipuler lorsque la fiabilité est supérieure à 0,9. Une échelle de grandeur logarithmique sur l'axe de la défiabilité permet d'avoir un meilleur rendu des fonctions de coût sur le domaine qui nous intéresse (défiabilité comprise entre 10^{-1} et 10^{-10}).
Sur la Figure 9.1, deux courbes sont affichées pour chaque fonction de coût : la première donne la forme générale du coût en fonction de la fiabilité du composant (R variant de 0 à 1), la seconde permet de visualiser le coût en fonction de la défiabilité du composant sur une échelle logarithmique.

La fonction élémentaire possède bien les trois propriétés fondamentales. Le paramètre *a* permet de faire passer la fonction par un point de référence. La Figure 9.1b montre que l'allure de la courbe n'est pas modifiée par la valeur du paramètre *a*.

En partant d'un point provenant du retour d'expérience (R_{Rex},C_{Rex}) et d'une fonction de coût passant par ce point, il est intéressant de connaître *l'intervalle de variation* de la fiabilité [R_{Min} , R_{Max}] (ou défiabilité) pour un coût variant entre $C_{Min}=C_{Rex}/\beta$ et $C_{Max}=C_{Rex}*\beta$, où β est un nombre généralement pris égal à 10.
Par exemple, pour passer par le point de défiabilité 10^{-4} et de coût 30, il faut fixer *a* à 0,003. Dans ce cas, le seuil intrinsèque de la défiabilité est fixé à 3×10^{-7} (pour un coût d'environ 10^{+4}) et l'intervalle de variation est de [10^{-3} ; 10^{-5}] pour une variation de coût de [3 ; 300].

La fonction élémentaire a une propriété géométrique intéressante. Lorsque la défiabilité d'un composant est multipliée par α, son coût est alors divisé par α. Cette fonction est utile lorsque la seule donnée connue pour un composant est son point de référence. Dans ce cas, l'allocation permettra d'explorer les solutions proches du point de référence.

9.2.3 FONCTIONS DE COUT DE LA LITTERATURE

La thèse de C.Elegbede [Ele00] propose (au chapitre 2.7) un panorama des principales fonctions de coût trouvées dans la littérature. Nous proposons dans ce paragraphe une analyse critique de ces fonctions de coût.
Le paramètre *a* est fixé à une valeur constante sur l'ensemble des courbes de ce chapitre, car on souhaite mettre en avant l'effet du paramètre *b* sur la fonction de coût.

La fonction de Breipohl [BJ]

Produit d'une exponentielle et d'une fonction rationnelle, la fonction de Breipohl est donnée par l'équation :

$$C_{Br}(R) = a \times \frac{1}{1-R} \times \exp\left(b \times (1-R)\right) \qquad [9\text{-}2]$$

b doit être inférieur ou égal à 1, pour que la fonction soit croissante.

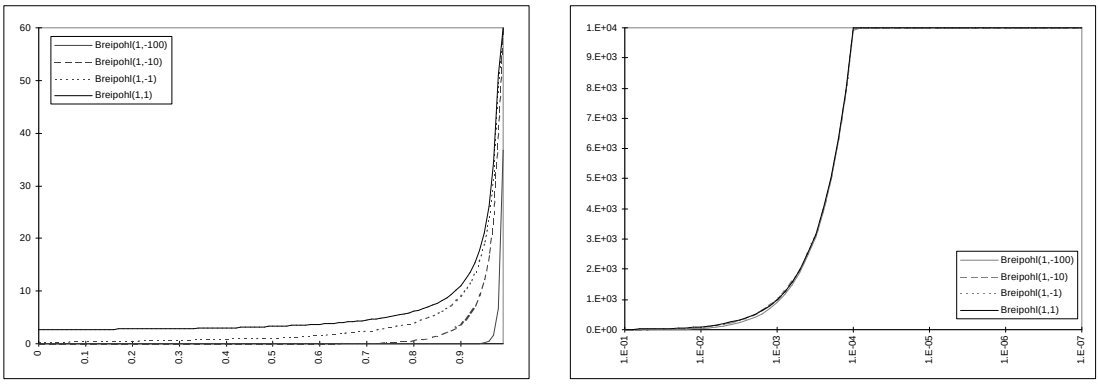


Figure 9.2 : Fonction de coût de Breipohl (a = 1 ; b = 1 ; -1 ; -10; -100)

Le paramètre *b* modifie principalement le début de la fonction (surtout pour *b* > -10).
Lorsque *R* tend vers 1, C_{Br} tend vers C_{El}.
En comparaison avec la fonction de coût élémentaire, l'effet du paramètre *b* permet de diminuer l'intervalle de variation et de reculer légèrement le seuil de fiabilité.

La fonction de Truelove [BJ]

Fonction rationnelle relativement simple, la fonction de Truelove est donnée par l'équation :

$$C_{Tr}(R) = a \times \frac{1}{(1-R)^b} \qquad [9\text{-}3]$$

où *b* est un réel strictement positif.

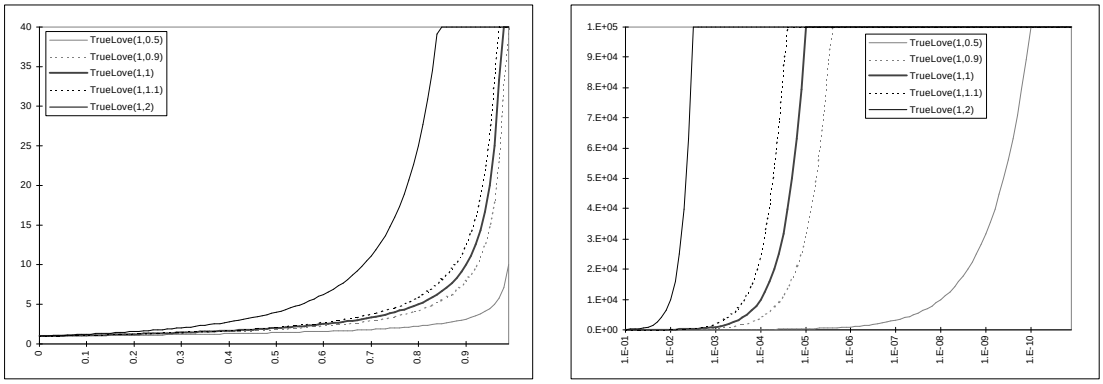


Figure 9.3 : Fonction de coût de Truelove (a = 1 ; b = 0.5 ; 0.9 ; 1 ; 1.1 ; 2)

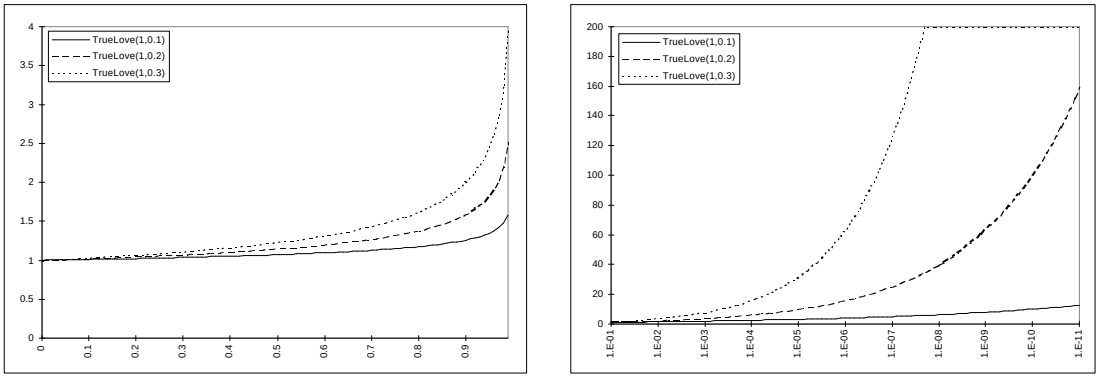


Figure 9.4 : Fonction de coût de Truelove (a = 1 ; b = 0.1 ; 0.2 ; 0.3)

Les Figure 9.3 et Figure 9.4 permettent de visualiser l'effet de *b* sur les allures de la fonction de coût de Truelove. L'intervalle de variation autour d'un point référence et le seuil de fiabilité peuvent être réglés beaucoup plus finement à l'aide de cette fonction.
Cette fonction peut passer de manière exacte par deux points de référence (R_1, P_1) et (R_2, P_2) en fixant les paramètres a et b à l'aide des équations :

$$a = C_1 \times \left(1 - R_1\right)^b = C_2 \times \left(1 - R_2\right)^b \qquad [9\text{-}4]$$

$$b = \log_{\frac{1-R_2}{1-R_1}} \left(\frac{C_1}{C_2}\right) = \frac{\ln\left(\frac{C_1}{C_2}\right)}{\ln\left(\frac{1-R_2}{1-R_1}\right)} \qquad [9\text{-}5]$$

Lorsque *b* est égal à 1, la fonction de Truelove est équivalente à la fonction élémentaire.

La fonction de Misra et al. [ML73]

Cette fonction est donnée par l'équation suivante :

$$C_{Mi}(R) = a \times \exp\left(\frac{b}{1-R}\right) \qquad [9\text{-}6]$$

pour laquelle b est un réel strictement positif.

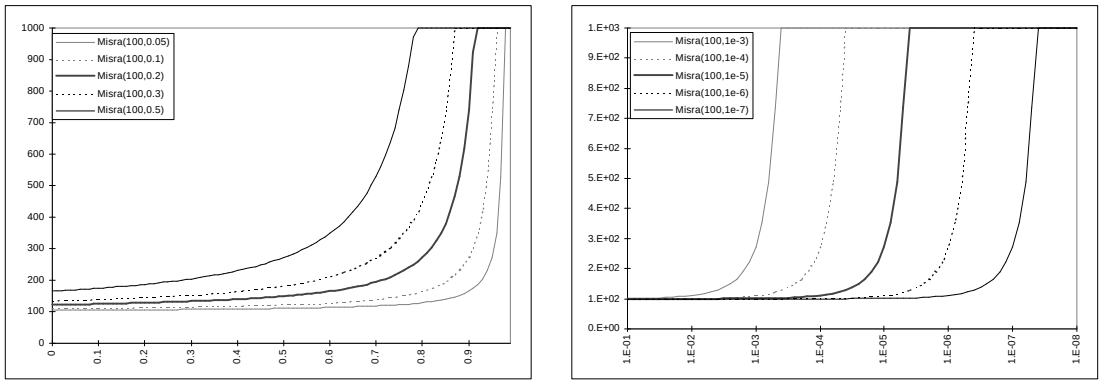


Figure 9.5 : Fonction de coût de Misra
(a = 100; b = 0.5 ; 0.3 ; 0.2 ; 0.1 ; 0.05 ; 10^{-3} ; 10^{-4} ; 10^{-5} ; 10^{-6} ; 10^{-7})

Le coût minimum d'un composant est supérieur ou égal au paramètre *a*. Pour un paramètre *a* fixé, le type d'évolution de cette fonction de coût ne dépend pas du paramètre *b*. Dans ce cas, ce dernier permet uniquement de fixer le seuil de fiabilité du composant. Ces différentes propriétés rendent la détermination des paramètres assez difficile.

En comparaison avec la fonction élémentaire, la pente sur une échelle logarithmique est plus faible à l'aide des fonctions de coût de Misra. En revanche, la prise en compte d'un composant de moindre qualité est malaisée avec cette fonction.

La fonction d'Aggarwall et al. [AGM75]

La fonction d'Aggarwall est définie par la fonction suivante

$$C_{Ag}(R) = a \times h(R) \times \tan\left(\frac{p}{2} \times R\right)$$

$$\begin{aligned} avec \quad & h(R) = 1 + R^a \quad si\ 0 \le a \le 1 \\ & h(R) = a \quad si\ 1 < a \le 2 \end{aligned} \qquad [9\text{-}7]$$

où a est compris entre 0 et 2 (inclus).

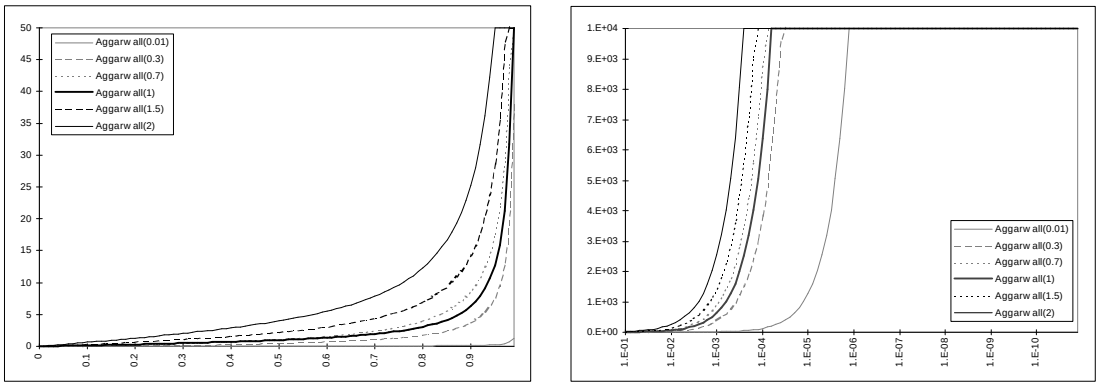


Figure 9.6 : Fonction de coût de Aggarwall (a = 0.01 ; 0.3 ; 0.7 ; 1 ;1.5 ; 2)

Cette fonction ne dispose que d'un seul paramètre. Il est difficile de la faire passer par un point de référence donné.

La fonction de Tillman et al.

Cette fonction est donnée par l'équation suivante :

$$C_{Ti}(R) = a \times R^b \qquad [9\text{-}8]$$

pour laquelle b est un réel strictement compris entre 0 et 1.

Cette fonction ne possède pas la troisième propriété des fonctions de coût. En effet, son coût tend vers l'asymptote de valeur a au voisinage de 1, quel que soit b (même pour b > 1).

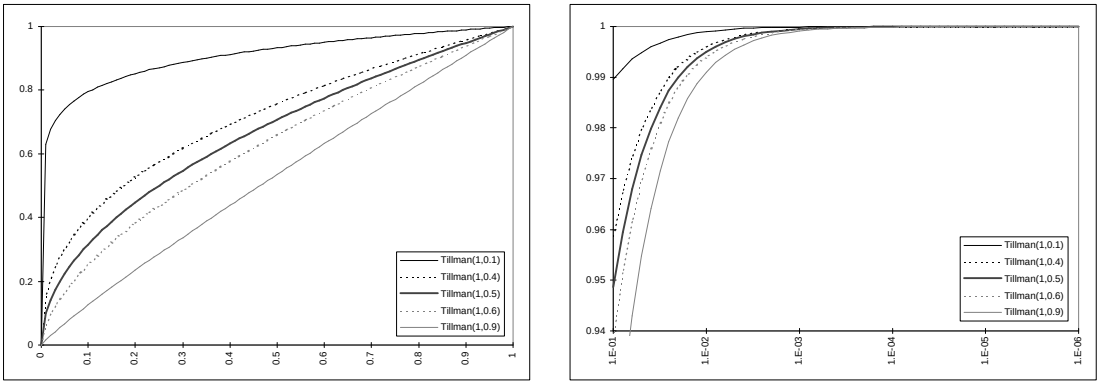


Figure 9.7 : Fonction de coût de Tillman (a = 1 ; b = 0.1 ; 0.4 ; 0.5 ; 0.6 ; 0.9)

Si b est supérieur à 1, on obtient une courbe ressemblant davantage aux autres fonctions de coût.

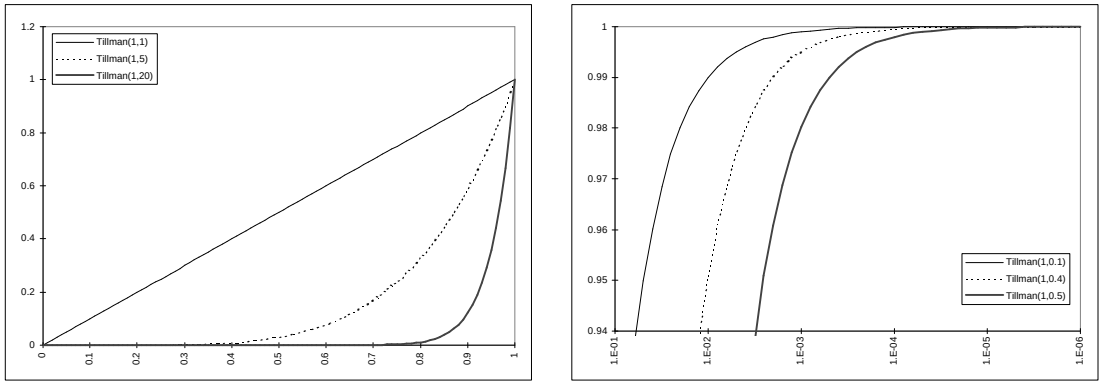


Figure 9.8 : Fonction de coût de "Tillman" (a = 1 ; b = 1 ; 5 ;20)

La fonction de Majundar et al.

Cette fonction est donnée par l'équation suivante :

$$C_{Ma}(R) = a \times \left(\frac{R}{1-R} \right)^{b} \qquad [9\text{-}9]$$

où b est un nombre réel strictement positif.

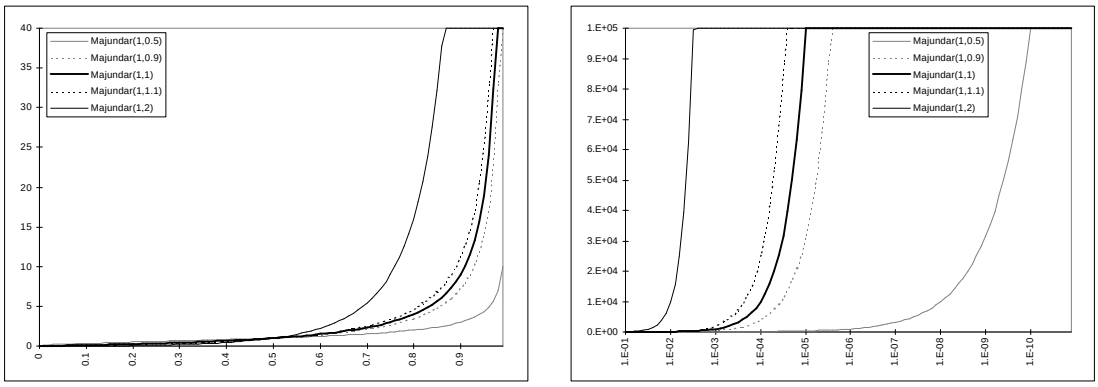


Figure 9.9 : Fonction de coût de Majundar (a = 1 ; b = 0.5 ; 0.9 ; 1 ; 1.1 ; 2)

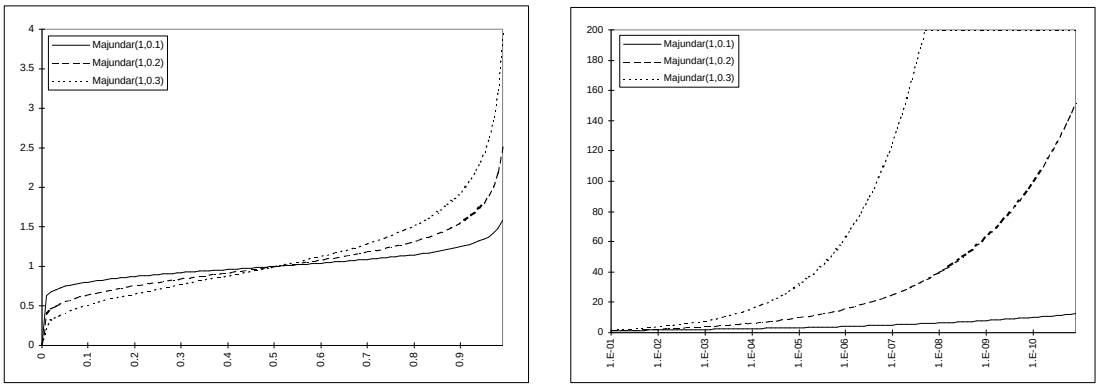


Figure 9.10 : Fonction de coût de Majundar (a = 1 ; b = 0.1 ; 0.2 ; 0.3)

Cette fonction est très proche de la fonction de Truelove, sauf en ce qui concerne l'intervalle de fiabilité [0 ; 0,5]. Son avantage est que le coût d'un composant ne fonctionnant jamais est de 0, alors qu'il est égal au paramètre a pour la fonction de Truelove.

Récapitulatif

Afin d'évaluer la richesse de paramétrage des fonctions de coût, nous proposons de les tester sur un exemple pratique. Considérons un composant ayant un point significatif connu ($Q = 10^{-4}$ pour un coût de 30). La limite technique de réalisation d'un composant de ce type est fixé à 10^{-8}. Supposons de plus que le coût total ne doit pas dépasser 1000. On peut donc considérer qu'un coût de 10 000 est un coût exhorbitant pouvant correspondre au seuil de fiabilité.

Les paramètres *a* et *b* sont calculés de façon à ce que les fonctions de coût passent toutes par le point de référence $\{10^{-4} ; 30\}$ et, dans la mesure du possible également par le seuil de fiabilité $\{10^{-8} , 10^{+4}\}$.

Fonction	b	a	C(0)	C(1)	Seuil intrinsèque	Intervalle de variation
Elémentaire	-	0,003	0,003	∞	3×10^{-7}	$[1\times10^{-3} ; 1\times10^{-5}]$
Breipohl 1	-10	0,003003	$1,36\times10^{-7}$	∞	$3,003\times10^{-7}$	$[0,99\times10^{-3}; 1,00\times10^{-5}]$
Breipohl 2	-100	0,00303015	0	∞	$3,03015\times10^{-7}$	$[0,92\times10^{-3}; 1,01\times10^{-5}]$
Breipohl 3	-10000	0,00815485	0	∞	$8,09\times10^{-7}$	$[2,42\times10^{-4}; 2,19\times10^{-5}]$
Truelove	0,63	0,09	0,09	∞	$9,79\times10^{-9}$ *	$[3,83\times10^{-3} ; 2,56\times10^{-6}]$
Misra 1	2×10^{-5}	24,56	24,56	∞	$3,33\times10^{-6}$	$[0 ; 7,99\times10^{-6}]$
Misra 2	5×10^{-6}	28,54	28,54	∞	$8,535\times10^{-7}$	$[0 ; 2,13\times10^{-6}]$
Misra 3	5×10^{-6}	28,54	28,54	∞	$1,03\times10^{-8}$ *	$[0 ; 2,61\times10^{-8}]$
Tillman	58088	10000	0	10000	0	$[1,4\times10^{-3} ; 6,04\times10^{-5}]$
Majundar	0,63	0,09	0	∞	$9,79\times10^{-9}$	$[3,83\times10^{-3} ; 2,56\times10^{-6}]$

Tableau 9-1 : Comparaison des différentes fonctions de coût

L'examen du tableau récapitulatif nous permet d'apprécier certaines caractéristiques des fonctions testées :
La fonction de Breipohl ne permet pas de repousser le seuil intrinsèque de manière significative et est confondue avec la fonction élémentaire.
La fonction de Misra donne des profils qui stagnent pendant trop longtemps et croissent trop rapidement. La différence de coût entre le composant de référence et un composant ne fonctionnant pas est de 5,44 (18,1 %) pour le premier jeux de paramètres et de 1,46 (4,9 %) pour le second.
En ce qui concerne la fonction de Tillman, *a* doit être fixé à 10000 (coût exorbitant lorsque *R* tend vers 1). Dans ce cas, pour passer par le point de référence $(30 ; 10^{-4})$, *b* doit être supérieur à 1 (58088), ce qui ne correspond pas à la définition de cette fonction de coût. L'intervalle de variation est alors inexploitable $[1,4\times10^{-3} ; 6,04\times10^{-5}]$.
La fonction de Majundar peut-être confondue avec celle de Truelove.

En conclusion, il ressort que :

- La fonction élémentaire, qui a une propriété géométrique simple (C'/R' = C"/R"), peut être utilisée aussi bien pour des méthodes paramétriques que pour des méthodes d'optimisation.
- La fonction de Truelove permet une bonne approximation pour les composants ayant un point de référence et un seuil technique de fiabilité. De plus, si le paramètre *b* de cette fonction vaut 1, cette fonction est égale à la fonction élémentaire.

Il est donc très difficile de justifier l'existence des autres fonctions de coût de la littérature, en tout cas dans le cadre de l'étude menée ici.

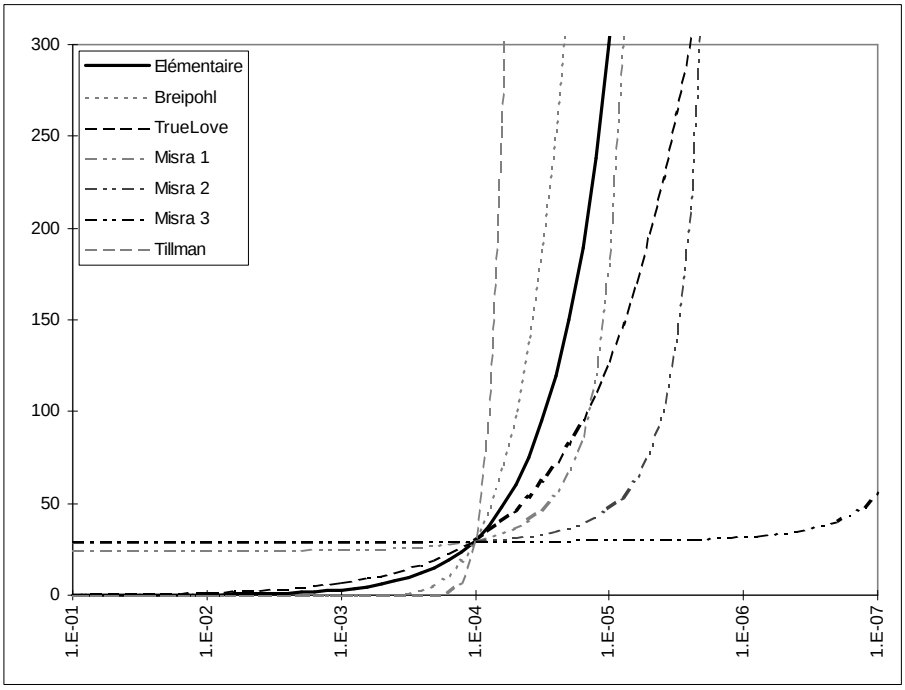


Figure 9.11 : Comparaison des différentes fonctions de coût (Coût entre 0 et 300)

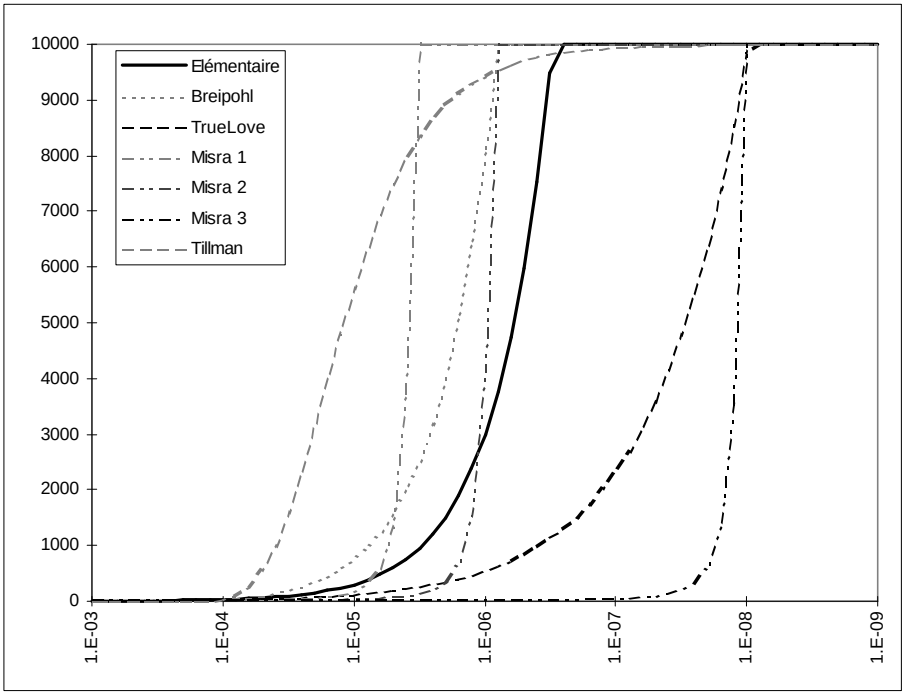


Figure 9.12 : Comparaison des différentes fonctions de coût (Coût entre 0 et 10000)

9.2.4 FONCTION DE COUT "EMPIRIQUE"

Lorsque l'on dispose d'un ensemble de points significatifs, il faut encore se demander comment évolue la fonction de coût entre ces points.
Prenons un exemple avec deux points de référence (composant 1 et composant 2), présentés sur la Figure 9.13. Il y a plusieurs interpolations naturelles possibles pour associer une fonction de coût passant par ces deux points :

- La première consiste à considérer une fonction de coût en escalier (interpolation rectangulaire). Pour une fiabilité inférieure ou égale à R1, le coût est égal à C1. Pour une fiabilité comprise entre R1 (exclu) et R2 (inclus), le coût est de C2, pour une fiabilité supérieure à R2, le coût est infini.
- La deuxième est de relier les deux points de référence par une droite (interpolation triangulaire). Un composant M ayant une fiabilité (R1+R2)/2 aura un coût de (C1+C2)/2.

La première méthode permet de faire un choix entre différents composants sur étagère. Elle force l'algorithme d'optimisation à choisir parmi les points proposés.
La seconde méthode reste à discuter.

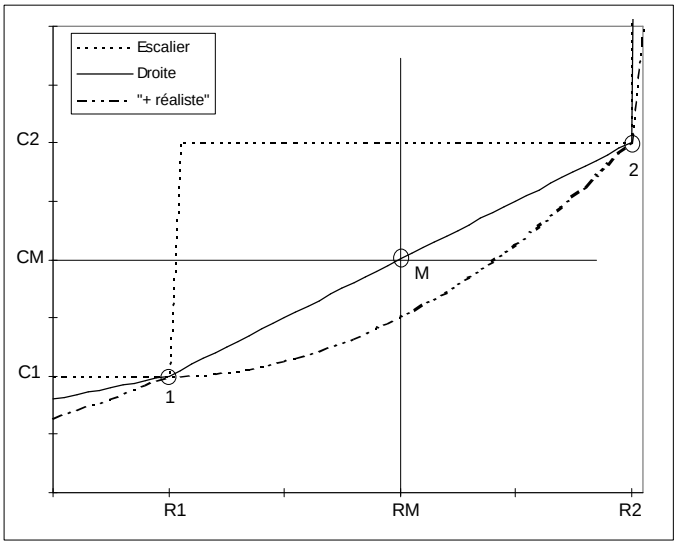


Figure 9.13 : Profil d'une fonction de coût passant par deux points

Intuitivement, l'effort pour passer du composant 1 au composant M doit être moins important que l'effort pour passer du composant M au composant 2.
Il est relativement aisé de faire un parallèle entre cette intuition et le principe du gain marginal décrit par Andrews et Moss dans [AM93]. Le gain marginal est mis en évidence pour un système de coût C et de fiabilité R ($R = 1-Q$) donné. Une manière de rendre plus fiable ce système est de le mettre en redondance. Pour une redondance d'ordre 2, le nouveau système coûte maintenant $2\times C$, sa fiabilité est de $1-Q^2$ (en négligeant la mise en œuvre de la redondance et les défaillances de cause commune). Pour une redondance d'ordre k, le système coûte $k\times C$, sa fiabilité est de $1-Q^k$. Le Tableau 9-2 est une application numérique pour $C = 10$ et $Q = 0{,}1$.

Redondance d'ordre	Coût	Défiabilité	Fiabilité	Gain en terme de fiabilité	Comparaison de la fiabilité
1	10	10^{-1}	0,9	-	-
2	20	10^{-2}	0,99	0.09	10 %
3	30	10^{-3}	0.999	9×10^{-3}	11 %
4	40	10^{-4}	0,9999	9×10^{-4}	11,1 %

Tableau 9-2 : Gain marginal pour un composant de coût C=10 et de défiabilité Q=0,1

Pour la même différence de coût, le gain en fiabilité diminue. Plus un composant est fiable, plus il faudra d'efforts pour améliorer de manière significative sa fiabilité. Ce phénomène est renforcé lors de la prise en compte des défaillances de cause commune.

Comme le montre la Figure 9.13, l'interpolation triangulaire ne permet pas de rendre compte du phénomène de gain marginal. Ces considérations, nous conduisent à proposer une troisième fonction d'interpolation : l'interpolation "creusée". Cette fonction est définie pour R compris entre R_1 et R_2,

$$\begin{aligned} & R_1 \le R \le R_2 \\ & \lim_{R \to R_1}(C(R)) = C_1 \\ & \lim_{R \to R_2}(C(R)) = C_2 \end{aligned} \qquad [9\text{-}10]$$

La fonction de coût suivante répond aux critères précédents :

$$\begin{aligned} & C(R) = C_1 + (C_2 - C_1) \times f(R) \\ & \text{avec } \lim_{R \to R_2}(f(R)) = 1 \text{ et } \lim_{R \to R_1}(f(R)) = 0 \end{aligned} \qquad [9\text{-}11]$$

La fonction sur la fiabilité suivante fait intervenir un paramètre utilisateur b.

$$f(R) = \left(\frac{R - R_1}{R_2 - R_1} \right)^b \qquad [9\text{-}12]$$

Cette dernière fonction n'est pas toujours définie pour R = R1 (lorsque b = 0).
L'interpolation proposée est donc la suivante :

$$C(R) = C_1 + (C_2 - C_1) \times \left(\frac{R - R_1}{R_2 - R_1} \right)^b \text{ avec } R_1 < R \le R_2 \qquad [9\text{-}13]$$

L'interpolation est rectangulaire lorsque b=0, triangulaire lorsque b=1, creusée lorsque b>1. La Figure 9.13 a été réalisée pour les trois valeurs de b suivantes (b=0, b=1 et b=2).

A partir d'un ensemble de points significatifs auquels ont été ajoutés les points (0 ; 0) (composant toujours en panne, de coût nul) et (1 ; ∞) (composant jamais défaillant, de coût infini), la fonction d'interpolation précédente permet de définir une fonction de coût "empirique", provenant d'un retour d'expérience.

La Figure 9.14 présente les interpolations pour différentes valeurs de b pour l'ensemble de points significatifs suivants donnés sous la forme (Q=1-R ; C) : (10^{-3} ; 10) , (10^{-4} ; 20) , (10^{-5} ; 35) , (10^{-6} ; 55).
Les deux points suivants ont été ajoutés : (1 ; 0) et (0 ; 10^{+6}), en supposant que le coût infini est fidèlement représenté par la valeur 10^{+6}. L'échelle logarithmique est nécessaire à la visualisation de la courbe.

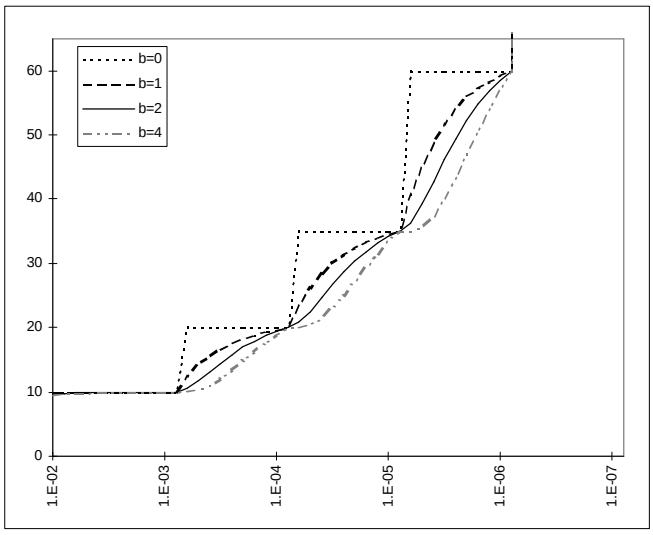


Figure 9.14 : Fonction de coût "empirique"

9.2.5 CONCLUSION

Parmi les fonctions de coût de la littérature, la fonction de Truelove, bien que relativement simple, correspond bien aux types de fonctions de coût souhaités dans les problèmes d'optimisation. Elle généralise la fonction de coût élémentaire.
La fonction de coût "empirique" est réalisée à l'aide d'une interpolation paramétrique permettant de prendre en compte l'idée du gain marginal.

Le choix entre la fonction de coût de Truelove et la fonction de coût empirique dépendra du nombre de points significatifs connus de l'utilisateur.

Information connue sur le composant ou sous-système	Fonction de coût à utiliser
Un point de référence	Fonction de coût élémentaire
Deux points de référence	Fonction de coût de Truelove
Plusieurs points de référence	Fonction de coût "empirique"

Tableau 9-3 : Fonctions de coût à prendre en compte

9.3 BIBLIOGRAPHIE

[AGM75] K.K. Aggarwal, J.S. Gupta and K.D. Misra. *A new heuristic criterion for solving a redundancy optimization problem*. IEEE Transaction on Reliability, (1)86-87, 1975.

[ML73] K. Misra and M. Ljubojevic. *Optimal reliability design of a system : A new look.* IEEE Transactions on Reliability No. 1, 1973, pp255-258

[BJ] S. Balaban and R.H. Jeffers. *The allocation of system reliability : Developpement of procedures for reliability allocation and testing.* Technical report, Directorate of operational support engineering, US air force.

[Ele00] C. Elegbede. *Contribution aux méthodes d'allocation d'exigences de fiabilité aux composants de systèmes.* PhD thesis, Université de Technologie de Compiègne, 2000

[AM93] J.D. Andrews and T.R. Moss. *Reliability and Risk Assessment.* John Wiley & Sons, 1993. ISBN 0-582-09615-4.

10. LE PROJET ALOES

Ce chapitre est organisé suivant l'ordre chronologique des actions qui ont été entreprises pour mener à bien le projet Aloès. Deux outils d'allocation (Algc-Sherloc et ARPO) ont servi de premières bases de réflexion. Ils sont décrits dans la section 10.1. Puis des discussions avec les partenaires du groupe Aloès ont permis de spécifier les fonctionnalités attendues de l'outil Aloès (présentées dans la section 10.2). A partir de ces fonctionnalités, nous avons proposé une organisation logicielle (section 10.3) et un langage de description des problèmes d'allocation (section 10.4). L'outil Aloès a alors été conçu et développé. Cet outil traite les problèmes d'optimisation grâce à trois algorithmes n'ayant pas les mêmes domaines d'application. Ils sont présentés dans la dernière section (10.5).

10.1 HISTORIQUE DU PROJET

Le projet Aloès résulte d'une collaboration entre différentes entreprises et des laboratoires universitaires, chacun d'eux ayant apporté ses préoccupations, ses moyens, ses connaissances et ses motivations.
Deux des partenaires industriels (Technicatome et EDF) avaient déjà une certaine expérience des problèmes d'allocation d'indisponibilité. Chacun d'eux avaient en effet développé et utilisé un outil d'allocation.
Les deux outils considérés (Algc-Sherloc [DMRT98b] pour Technicatome et ARPO [BB97a] pour EDF) supposent que la défaillance du système considéré est modélisée à l'aide d'un arbre de défaillance. Ils sont tous les deux composés :

- d'un cœur de calcul autonome qui encapsule Aralia afin de traiter les arbres de défaillance du modèle ;
- d'une interface graphique permettant de définir le problème, de lancer et de suivre les calculs.

La différence essentielle entre Arpo et Algc/Sherloc est que l'espace des variables de décision d'Arpo est continu, alors que celui d' Algc/Sherloc est discret.
Les études menées à Technicatome (approche discrète) et à EDF (approche continue) ont donné des résultats satisfaisants.

10.1.1 PROJET ALGC/SHERLOC

Algc/Sherloc formalise le problème de l'allocation d'exigence de la façon suivante : on considère que la probabilité d'occurrence d'un événement de base *e* dépend d'un paramètre ρ_e. ρ_e est la probabilité elle-même, si *e* suit une loi de probabilité constante, ou le taux de défaillance, si *e* suit une loi de probabilité exponentielle. On considère de plus que ρ_e varie de façon discrète et décroissante en fonction du coût c_e du composant *e*. Cette notion de coût est bien entendu très abstraite et aucune signification particulière ne lui est imposée. Une fonction coût/fiabilité est donc associée à chaque composant, comme le représente la Figure 10.1.

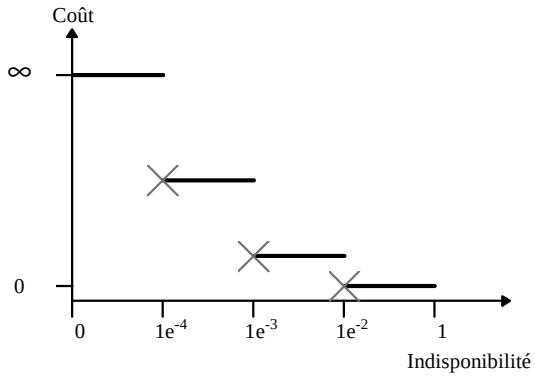


Figure 10.1 : Une fonction coût/fiabilité est associée à chaque composant.

On considère enfin que le coût de l'installation (de l'événement-sommet) est la somme des coûts des événements de base. Le problème est donc de minimiser la probabilité de l'événement-sommet en respectant

une contrainte sur le coût global ou, inversement, de minimiser le coût global en respectant une contrainte sur la probabilité de l'événement-sommet.
Deux algorithmes ont été mis en œuvre pour résoudre ce problème d'optimisation : un algorithme de séparation/évaluation (Branch & Bound) et une méthode stochastique de réparation locale. Les deux algorithmes nécessitent une fonction $fc(s_1,s_2)$ qui compare deux solutions potentielles. Les fonctions *fc* que nous avons utilisées améliorent la disponibilité jusqu'à un seuil prédéfini, puis améliorent le coût lorsque ce seuil est atteint.
L'outil Algc/Sherloc et les applications qui en ont été faites par Technicatome sont décrits plus en détail dans les articles [MRT97, DMRT98a, DMRT98b, DMRT98c].

10.1.2 PROJET ARPO

ARPO (Allocation Répartie par Pondération ou Optimisation) propose deux méthodes d'allocation :

- La méthode par pondération (présentée dans le paragraphe 8.1.2) a l'avantage de ne demander que peu d'informations sur le système considéré (l'arbre de défaillance décrivant la défaillance principale du système et les indisponibilités des différents composants du système provenant d'un retour d'expérience ou d'un jugement d'expert). Elle permet un calcul rapide et souvent satisfaisant.
- La méthode par optimisation dite de minimisation d'effort (AME) est beaucoup plus générale. Elle cherche à minimiser une fonction d'effort sur le système (qui peut être son coût d'acquisition ou de possession) en prenant en compte des contraintes sur les paramètres - variables de décision de l'étude, principalement indisponibilité, taux de défaillance ou de réparation d'un composant, et des contraintes sur l'indisponibilité et/ou la sécurité du système. Elle est formalisée de la manière suivante :
  Soit $X = (x_1, ..., x_N)$ le vecteur des variables de décisions du problème de taille *N*.
  Minimiser [P 10-1]
  $$F = E_S(X)$$
  Sous les contraintes
  $$C_j(X) \le B_j \qquad \forall j = 1..K$$
  $$\min_j \le x_j \le \max_j, \qquad \forall j = 1,\ldots,N$$
  où
  C_j est la probabilité d'occurrence d'un événement redouté du système. Il peut s'agir de son indisponibilité ou d'étude de sécurité sur des aspects de l'installation.
  B_j est le seuil de probabilité à respecter pour la contrainte j.
  E_S est la fonction d'effort du problème. Elle est en général définie comme la somme de efforts associés à chaque composant du système. La fonction d'effort la plus utilisée est celle de Truelove (Cf. paragraphe 9.2.3)
  Il s'agit donc d'une instance du problème P8-1, où le coût de l'installation est à minimiser et où il existe plus d'une contrainte de fiabilité sur le système.
  Ce problème d'optimisation est résolu à l'aide d'un algorithme du "Simplex" (Cf. paragraphe 10.5.4). Cette méthode donne souvent des solutions satisfaisantes.

L'outil ARPO et les applications qui en ont été faites par EDF, sont décrits plus en détail dans les articles [BB97a, BB97b].

10.2 SPECIFICATION DU PROJET ALOES

Un des objectifs du projet Aloès est d'imposer un minimum de limites de modélisation aux ingénieurs chargés de la conception d'un système, c'est-à-dire ceux qui sont censés mettre en œuvre les méthodes d'optimisation. Les propositions d'optimisation abondent dans la littérature et imposent avant tout des contraintes sur les modèles considérés et sur la manière dont les ingénieurs devront formaliser leurs problèmes. Les industriels de ce projet souhaitaient, avant tout, un cadre assez général permettant de prendre en compte un large spectre d'applications. Pour des raisons d'ingénierie, les spécificités des outils historiques du projet (Arpo et Algc/Sherloc) étaient, bien entendu, à prendre en considération.

Nous avons donc proposé un langage de description des problèmes d'allocation. Ce langage devait permettre de :

- Considérer indifféremment des variables de décision discrètes ou continues ;
- Définir les grandeurs à optimiser en fonction des variables de décision en utilisant les opérateurs mathématiques usuels ;
- Modifier d'autres paramètres de fiabilité que l'indisponibilité du composant, comme les taux de défaillance ou de réparation, les périodes de maintenance préventive, ...
- Prendre en compte des modèles de fiabilité utilisés dans les études de sûreté de fonctionnement comme les arbres de défaillance, les diagrammes ou les réseaux de fiabilité, les graphes de Markov, ...
- Considérer des variantes d'architecture comme la redondance.

10.2.1 VARIABLES DE DECISION CONTINUES OU DISCRETES

La question du domaine des variables de décision a été source de nombreuses discussions dans le groupe Aloès. Faut-il ne considérer que des variables discrètes ? Les variables continues apportent-elles une richesse de modélisation supplémentaire ? Quelques éléments de réponse sont présentés dans ce paragraphe.

Les méthodes d'optimisation prenant en compte des grandeurs fiabilistes ne considèrent en général que des variables de décision discrètes. Le niveau de redondance d'un équipement est décrit par des variables de décision discrètes.
En phase de conception avancée ou de modification de l'installation, elles permettent de bien prendre en considération l'idée de composant sur étagère. Un composant sur étagère est un équipement qui répond aux critères fonctionnels exigés pour le composant, mais qui existe déjà sur le marché et dont les grandeurs (coût d'acquisition, de maintenance, taux de défaillance, ...) sont connues. Si le niveau de détail d'une étude d'optimisation est l'équipement, il est logique de se référer aux produits du marché. Dans le cas contraire, l'optimisation pourrait fournir comme solution un composant ayant un taux de défaillance éloigné de tous les composants sur étagère disponibles. Il faudrait alors, pour être en adéquation avec l'optimisation, le faire construire. Or, le coût de développement d'un nouveau composant est nettement plus élevé que celui d'un composant du marché.

La planification de la maintenance préventive peut être représentée soit par des variables de décision continues, soit par des variables de décision discrètes. Ce choix se fera suivant le type du système étudié.
Dans certaines installations de production, une maintenance préventive suppose l'arrêt de la chaîne de production. Il convient alors de la planifier afin qu'elle n'engendre pas de diminution de productivité. Une méthode mise en œuvre consiste à planifier les maintenances préventives pendant les vacances, c'est-à-dire environ tous les quatre mois (Noël, Pâques, Eté). La périodicité de la maintenance pour un composant donné est alors fonction d'une variable de décision entière multipliée par la période de référence (ici, quatre mois).
D'autres installations permettent certaines maintenances préventives en fonctionnement. La périodicité de la maintenance préventive d'un composant est alors soumise à beaucoup moins de contraintes et peut être égale à une variable de décision continue.

Les variables de décision continues permettent de faire une optimisation sur des systèmes en conception préliminaire de manière plus réaliste. Lors des premières phases de conception, le niveau de détail de l'analyse est rarement celui des composants d'un système, mais plutôt celui des sous-systèmes constituant l'installation. Les variables de décision continues représenteront par exemple le taux de défaillance associé à chacun des sous-systèmes. Des jugements d'experts et le retour d'expérience permettent d'associer un coût, ou plutôt un effort à fournir, en fonction du taux de défaillance.
Dans une optimisation prenant en compte la maintenance, les taux de réparation associés aux défaillances des équipements sont plus facilement représentables par des variables de décision continues. Il est possible de jouer sur les moyens de maintenance à mettre en œuvre tels que l'accessibilité, le nombre de pièces de rechange en stock, l'assistance au diagnostic ..., sans qu'il soit possible de les modéliser exactement. En revanche, à partir d'un taux de réparation supposé, il est possible de faire des recommandations afin de l'améliorer d'un facteur donné. Les équipes de conception et de maintenance doivent alors trouver des solutions techniques permettant de mettre en œuvre cet objectif.

En conclusion, le choix d'utiliser exclusivement des variables de décision continues ou discrètes dépend du

système et de l'étude à effectuer. Elles sont complémentaires et permettent de capter des phénomènes différents. C'est pour cette raison que nous avons intégré ces deux types de variables de décision dans nos modèles.

Pouvoir modéliser un problème en prenant en compte des variables de décision discrètes et/ou continues influe directement sur la mise en œuvre d'algorithmes de résolution permettant de trouver un optimum.

10.2.2 PRISE EN COMPTE DE DIFFERENTS MODELES FIABILISTES

Les diagrammes de fiabilité simples correspondant aux systèmes série, parallèle, série-parallèle, parallèle-série, ... doivent être directement intégrés dans Aloès, étant donné que la fiabilité ou la disponibilité de ces systèmes sont calculables à l'aide d'une formule littérale.
L'utilisation de modèles plus complexes nous oblige à utiliser des logiciels permettant de les évaluer.
Ainsi ARPO et Algc/Sherloc sont couplés à Aralia pour le calcul de la disponibilité à partir d'un arbre de défaillance représentant un des événements redoutés pris en considération. Ren et Dugan [RD98] intègrent directement leur méthodologie d'optimisation dans DifTree : logiciel de traitement et de quantification d'arbres de défaillance statiques et dynamiques. La référence [ACNT99] utilise un module Excel (GenOpti) proposant une méthode assez générale d'optimisation (pas seulement limitée à l'allocation de fiabilité). Ce module peut être couplé à d'autres modules Excel permettant d'évaluer les arbres de défaillance ou de traiter les modèles markoviens.
Le cadre de Aloès est limité aux méthodes d'allocation d'exigences. Il n'était pas question de réaliser des logiciels (ou modules) de traitement et de quantification des modèles de la sûreté de fonctionnement. Prendre en considération ces modèles devra donc se faire en utilisant des logiciels ou modules existants et en les couplant d'une manière ou d'une autre à Aloès.
Une des spécifications du projet était, bien entendu, de pouvoir utiliser la rapidité de calcul d'Aralia et la facilité de conceptualisation d'une allocation sur un diagramme ou un réseau de fiabilité à l'aide de Réséda. Mais est rapidement apparu l'intérêt de ne pas se limiter qu'aux modèles booléens d'analyse des risques. Par exemple, dans l'avenir, un module markovien permettra de modéliser une politique de maintenance réaliste, des dépendances temporelles entre composants de base, des reconfigurations dynamiques du système ...

10.2.3 PRISE EN COMPTE DES CHOIX D'ARCHITECTURE

Les choix d'architecture dans les problèmes d'optimisation sont généralement limités à des problèmes de redondance. Si les composants de cette redondance sont indépendants du reste du système, il est possible de calculer littéralement les grandeurs fiabilistes de l'équipement à partir des variables de décision représentant le niveau de redondance et des grandeurs fiabilistes associées aux composants.

La référence [PK00] recense tous les types de redondance pris en compte habituellement dans les problèmes d'optimisation :

1. L'équipement est composé de *n* composants identiques de fiabilité r_c en redondance active totale :

$$R_{Eq}(n) = 1-\left(1-r_c\right)^n \qquad [10\text{-}1]$$

2. L'équipement est composé de *n* composants identiques de fiabilité r_c en redondance active *k*/*n* (au moins *k* composants doivent fonctionner) :

$$R_{Eq}(n) = \sum_{i=k}^{n}\left(\binom{n}{i}\times {r_c}^i \times\left(1-r_c\right)^{n-i}\right) \qquad [10\text{-}2]$$

3. L'équipement est composé de n composants identiques de taux de défaillance λ_c en attente (redondance passive d'ordre k avec un système de commutation fiable)

$$R_{Eq}(n,t) = \sum_{i=1}^{n}\frac{\left(l_c\times t\right)^{i-1}\times\exp\left(-l_c\times t\right)}{(i-1)!} \qquad [10\text{-}3]$$

Ces composants en redondance sont alors représentés dans les modèles booléens d'analyse des risques par un seul événement élémentaire. Les données, qui varient en fonction du choix fait, sont la loi de probabilité associée à cet événement élémentaire et d'autres paramètres tels que le coût, l'indicateur de performance …

L'avantage des problèmes se situant à ce niveau est que le modèle n'a pas à prendre en compte ces variantes d'architecture, car les calculs peuvent être réalisés au préalable au sein d'Aloès.

Il en va autrement lorsque les composants en redondance ne sont plus indépendants du reste du système étudié. Choisir entre différentes architectures, dans le cadre général est *a priori* un problème beaucoup plus complexe à résoudre. En effet, qui dit plusieurs architectures du système, dit plusieurs modèles fiabilistes représentant ces architectures.

Dans certains modèles, il est possible de factoriser les différentes architectures au sein d'un seul modèle. C'est le cas des arbres de défaillance qui se servent des événements de configuration (constantes nommées, mis "à vrai" ou "à faux" suivant le calcul) pour valider ou invalider une ou des branches de l'arbre. Les choix d'architecture sont modélisés par des événements de configuration dont la valeur détermine l'architecture. Ces événements et les alternatives qu'ils offrent sont présents dans l'arbre qui décrit, de la sorte, toutes les architectures possibles. Les calculs de probabilité deviennent des calculs de probabilités conditionnelles (qui fixent la valeur des événements de configuration). La faisabilité d'une telle approche a été démontrée dans [AP97].
Ren et Dugan [RD98] propose l'ajout de 2 portes dans le formalisme des arbres de défaillance : la porte CHO (comme choice) pour définir un choix entre différentes architectures et la porte RED (comme redundance) pour définir une possibilité de redondance active sur un sous-système. Ces portes peuvent être compilées en "Ou" de "Et" ou en "Et" de "Ou" avec des événements de configuration adéquats afin de pouvoir traiter les problèmes d'optimisation. L'intérêt est que les redondances et les choix d'architecture peuvent être fait au niveau d'un sous-système et pas seulement au niveau des composants élémentaires. Si ces sous-systèmes partagent des événements élémentaires, il n'est plus possible de faire un calcul préalable.

Les réseaux de fiabilité étant compilés en formules booléennes, la technique des événements de configuration peut être appliquée. On peut, par exemple associer une garde à chaque arête et à chaque nœud précisant la liste des architectures dans lesquelles le composant est censé fonctionner. Lors de la traduction du modèle, cette liste est prise en considération afin d'ajouter les événements de configuration représentant les choix d'architecture.

Comme l'a souligné Pierre-Etienne Labeau, la modélisation de variantes d'architectures par graphes de Markov n'induit pas nécessairement la création d'autant de modèles différents que d'architectures possibles.
"De la même manière que des événements de configuration peuvent être introduits dans les arbres de défaillance, on peut envisager d'avoir un graphe de Markov général (pour une catégorie de problèmes), dans lequel certains arcs seraient valués par le produit du taux de transition et d'une variable de décision binaire, valant 1 uniquement lorsque la transition modélisée par l'arc est bien possible dans la variante. Le couplage de ce graphe élargi avec un algorithme d'agrégation exacte des états, une fois la valeur des variables de décision fixée, réduirait sans doute significativement le graphe élargi de départ. Cette solution n'est peut-être pas la plus efficace, mais elle est certainement réalisable." (P.E. Labeau)
Une autre solution serait de disposer d'un outil de description de systèmes permettant de modéliser des choix d'architecture et pouvant traduire la description du système en graphes de Markov.

10.2.4 MONOTONIE

Un langage ayant de telles exigences risquent de conduire à des problèmes très compliqués à résoudre par les méthodes et algorithmes existants.
Le langage peut induire des problèmes d'optimisation. Citons en deux :

- [Che92] a montré que pour un système k-sur-n/série avec prise en compte de la redondance (discret) et de l'amélioration de la fiabilité des composants (continu) le problème était NP-Difficile. Dans notre cas, la difficulté est encore plus grande du fait que les modèles considérés ne se réduisent pas nécessairement à des systèmes k-sur-n/série.
- Si, de plus, les grandeurs à optimiser (ou sous contraintes) ne sont pas monotones en fonction des variables de décision, le problème s'avère encore plus ardu.

De nombreux algorithmes fonctionnent de manière plus efficace lorsqu'ils ont des informations sur les relations existant entre les grandeurs à optimiser et les variables de décision.

Dans un premier temps, une solution serait de demander à l'utilisateur les relations de dépendance entre les grandeurs et les variables de décision. Soit G une grandeur à optimiser et Vi une variable de décision. Il convient de savoir si G est fonction de Vi, si elle est monotone ou non en fonction de Vi, et si elle est croissante ou décroissante en fonction de Vi.
Un algorithme fixant cette relation de dépendance peut sans doute être mis en place. Une des difficultés provient du fait que certaines données ne sont pas connues par Aloès (module externe).
Dans la majorité des cas, ces relations de dépendance sont connues par celui qui réalise le modèle. Si nous cherchons à optimiser un système cohérent (représenté par une fonction booléenne monotone), nous avons la certitude que plus les composants seront fiables, plus le système sera fiable. Si la fiabilité des composants est égale à la valeur d'une variable de décision, la fiabilité du système est croissante en fonction de chaque variable de décision du système.
Les fonctions ne sont monotones que dans certains cas. Par exemple, lorsque nous essayons de tester pour un système plusieurs architectures assez différentes, le coût et la fiabilité de ce système dépendra des variables de décision prenant en compte les variantes d'architecture et des coûts/fiabilités des composants de base de chaque architecture. Dans ce cas, nous ne pouvons pas dire *a priori* si le coût ou la fiabilité du système est monotone en fonction de la variable de décision représentant les variantes d'architecture.
Cette relation de dépendance entre les contraintes et les variables de décision pourra être spécifiée à l'aide du langage. Cela permettra aux algorithmes de choisir des heuristiques différentes en fonction de ces informations de dépendance.

10.3 ARCHITECTURE LOGICIEL DE ALOES

Une des idées fortes du projet Aloès est de considérer des moteurs de calcul permettant l'évaluation probabiliste des modèles de fiabilité comme des modules extérieurs à Aloès. Cette idée a eu des répercutions :

- sur le langage Aloès : Il faut pouvoir inclure, à partir d'une construction linguistique, des données propres aux modules extérieurs et définir une fonction générique permettant d'échanger des informations entre Aloès et les moteurs de calcul. L'échange d'informations doit se faire dans les deux sens. Il peut s'agir de la modification d'une donnée du modèle fiabiliste ou du résultat d'un calcul.
- sur l'architecture du logiciel : Une interface générique a donc été spécifiée. Elle doit permettre de coupler un logiciel de traitement de données, au sens large, à Aloès. Aloès ne communiquant qu'à travers cette interface générique, il ne connaît pas la nature des logiciels avec lequel il coopère. Inversement, les logiciels externes ne connaissent pas plus Aloès, car ils ne font que répondre à des requêtes de l'interface. En fait, seule l'interface doit connaître les deux parties : Aloès et le moteur de calcul externe.

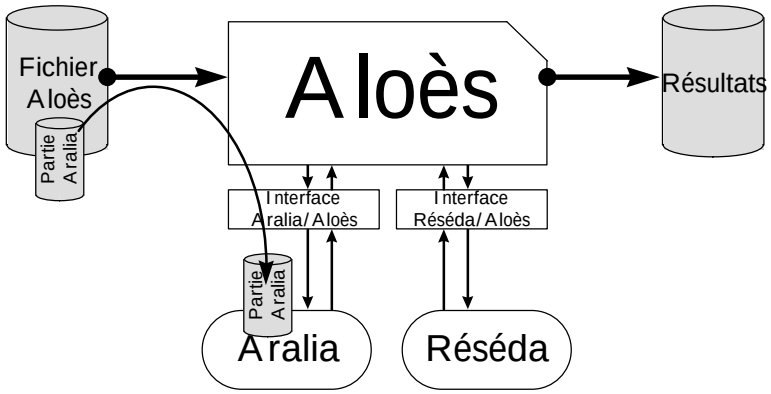


Figure 10.2 : Couplage de modules externes à Aloès.

De cette manière, l'ajout d'autres moteurs de calcul peut se faire sans difficulté.

Ce n'est pas une surprise de préciser que les moteurs de calcul considérés dans un premier temps sont :

- Aralia, un logiciel de traitements qualitatif et quantitatif de formules booléennes probabilisées, a été présenté dans l'introduction de cette thèse. Nous développons ici les caractéristiques intéressantes de cette outil dans le cadre d'une allocation. Il permet de traiter directement les arbres de défaillance. L'intérêt qu'il

présente est qu'il s'appuie sur un paquetage BDD. Nous avons vu au chapitre 2.6 que grâce à cette structure de données, il n'était pas nécessaire de calculer les coupes minimales du système pour calculer la probabilité de l'événement redouté. De plus, une fois le diagramme binaire de décision codant la formule booléenne construit, le calcul de la probabilité de cette formule, c'est-à-dire la probabilité d'occurrence d'un événement redouté, se fait en temps linéaire par rapport la taille du BDD. Cette propriété nous intéresse particulièrement, car dans un problème d'optimisation le nombre de calculs inhérents aux solutions potentielles peut être très important. Si le calcul d'une des grandeurs (celle à optimiser ou une des contraintes) prend du temps, le temps de résolution du problème d'optimisation en prendra d'autant plus.

- Réséda, un compilateur de réseaux de fiabilité vers les formules booléennes, qui, à partir d'un réseau, de ses nœuds source et cible et de directives de compilation, génère un fichier au format Aralia. Ce dernier traite alors ces données de la même manière qu'un arbre de défaillance ayant comme événement redouté, "la cible n'est plus alimentée". L'intérêt de ce module est d'ordre conceptuel. Les diagrammes et les réseaux de fiabilité sont des modèles sur lesquels l'allocation est beaucoup plus "naturelle".

10.4 LE LANGAGE ALOES

Le langage Aloès comporte trois types d'objets, chacun d'eux correspondant à une structure syntaxique :

- Les modèles qui regroupent toutes les données décrivant le système étudié : les variables de décision, les variables intermédiaires, ...
- Les problèmes qui correspondent à une question spécifique que l'on se pose sur un modèle donné.
- Les affectations qui correspondent à une valuation des variables de décision du modèle.

Ces trois types d'objets sont définis à l'aide d'un identificateur, afin de pouvoir y faire référence.

10.4.1 LES MODELES

Un modèle est composé de cinq types de données différents :

1. Des variables, dites de décision, dont le domaine peut être discret ou continu. Ces variables servent à décrire les choix potentiels.
2. Des variables, dites intermédiaires, dont les valeurs sont des nombres réels qui sont entièrement déterminés par la valeur des variables de décision. Les paramètres des lois de probabilités, les coûts, les indicateurs de performances …, sont décrits par de telles variables.
3. Des constantes.
4. Des équations qui fixent la valeur des variables intermédiaires en fonction de celles des variables de décision, éventuellement de celles d'autres variables intermédiaires et de constantes.
5. Des données spécifiques à un module de calcul extérieur. Les arbres de défaillance (ou les réseaux de fiabilité), qui décrivent les événements redoutés associés au système étudié, sont inclus dans de telles rubriques.

Une des idées du langage est donc de définir les codes de calcul spécifiques (du type Aralia) comme des modules externes à Aloès. Les données propres à ces modules (comme la fonction booléenne pour Aralia, le réseau de fiabilité, ...) sont incluses dans le modèle Aloès et un opérateur générique fait l'interface entre Aloès et ces modules externes.

Variables de décision

Les variables de décision correspondent en général aux choix technologiques ou organisationnels que l'on désire prendre en compte, comme les variantes d'architecture (redondances incluses), le taux de défaillance d'un composant, la périodicité de tests dans une politique de maintenance, le nombre de réparateurs, un critère de qualité d'un composant, ... Leurs domaines sont des intervalles entiers ou réels.
Elles sont déclarées de la façon suivante :

```
decision int var1 = [1 , 3] ;
float min2 = 1e-7;
float max2 = 0.5;
decision float var2 = [min2, max2] ;
```

La variable de décision var1 est un entier de domaine [1,3]. Elle peut refléter le niveau de qualité d'un composant :

- 1 : bas de gamme (pas cher)
- 2 : standard (un peu plus cher)
- 3 : haut de gamme (très cher)

La variable de décision var2 correspond à une plage de probabilités d'occurrence pour un événement élémentaire du système étudié. C'est une variable réelle pouvant prendre valeur dans l'intervalle [10^{-7} ; 0.5]. La borne inférieure représente dans ce cas, une limite de faisabilité. Le composant ne pouvant techniquement pas avoir une indisponibilité inférieure à 10^{-7}.
Les bornes d'une variable de décision sont obligatoires et nécessairement des constantes du modèle.

Variables intermédiaires et équations

Les variables intermédiaires sont des nombres réels. Leurs valeurs doivent être entièrement déterminées par la valeur des variables de décision.
L'indisponibilité ou le taux de défaillance d'un événement élémentaire peut être représenté par une variable intermédiaire. Le coût, les indicateurs de performance, la probabilité d'un événement redouté du système... sont décrits par de telles variables.
Les variables intermédiaires sont définies à l'aide d'équations du type $x = F$, où x est une variable intermédiaire et F est une formule dépendant des variables de décision, d'autres variables intermédiaires ou de constantes.
Les opérateurs arithmétiques (addition, soustraction, multiplication, division, modulo), logiques (et, ou, non), d'égalité (égal, différent) et d'inégalité (supérieur, inférieur, ...) ainsi que les fonctions mathématiques usuelles (exponentielle, logarithmique, puissance, ...) peuvent être utilisés dans ces équations.
De plus, l'opérateur **`switch`** permet de faire des sélections sur des variables de décision ou intermédiaires. Dans l'exemple suivant, l'opérateur **`switch`** renvoie différentes valeurs selon la qualité du composant décrite par la valeur de la variable de décision vd1.

```
float lbd1 := switch vd1 {
        1 : 1e-3,
        2 : 1e-4,
        3 : 1e-5
        };
```

Le test d'appartenance à un intervalle pour l'opérateur **`switch`** permet de prendre en compte les fonctions constantes par morceaux. Ainsi l'exemple suivant définit le coût du composant rattaché à vd2 par une fonction en escalier. Le test d'appartenance **`default`** renvoie toujours "vrai".

```
float cost2 := switch vd2 {
        [0.5 , 1] : 1,
        [0.2 , 0.5] : 5,
        [0.01 , 0.2] : 50,
        default : 1000
        } ;
```

Modules extérieurs

Les modèles de fiabilité, permettant le calcul de grandeurs de la Sûreté de Fonctionnement comme la disponibilité, la fiabilité, ..., sont intégrés à Aloès en tant que modules externes. Deux modules sont actuellement accessibles au sein d'Aloès : le module Aralia pour traiter les formules booléennes probabilisées, c'est-à-dire les arbres de défaillance et le module Réséda spécialisé dans le traitement de réseaux de fiabilité.

Aloès communique avec ces modules externes par deux moyens :

- La structure du modèle de fiabilité est définie sous la forme d'un bloc de données dont Aloès ignore tout. Elle est envoyée directement aux modules externes
- Une fonction générique est utilisée afin de relier les variables de décision aux données propres du module externe et de récupérer les résultats d'un calcul. Cette fonction prend comme paramètres une liste de chaînes de caractères, de constantes et de variables de décision. La signification et la syntaxe des paramètres dépendent entièrement du module externe. Ces fonctions retournent par défaut une valeur réelle lorsqu'elles sont évaluées.

```
aralia {{
..r := (g | f);
  g := (a & b);
  f := (c & d);
}} ;

aralia("law", "a", "exponential", lbd1);
float Q = aralia("Pr", "r", 10);
```

L'exemple ci-dessus associe au modèle en cours un arbre de défaillance à la syntaxe Aralia. Il modifie la loi associée à l'événement élémentaire *a* lorsque le paramètre *lbd1* change et déclare une grandeur *Q* qui s'avère être la probabilité (*Pr*) de la variable *r* au temps *t* = 10.

Relations de dépendance

Les relations de dépendance permettant d'améliorer généralement la rapidité des algorithmes sont fixées à l'aide de la syntaxe :

```
dependency G V0 + ; // G est monotone croissante en fonction de V0
dependency G V1 - ; // G est monotone décroissante en fonction de V1
dependency G V2 ? ; // G est non monotone en fonction de V2 (valeur par défaut)
dependency G V3 ! ; // G n'est pas fonction de V3
```

Exemple de modèle

Le problème à modéliser est le suivant :

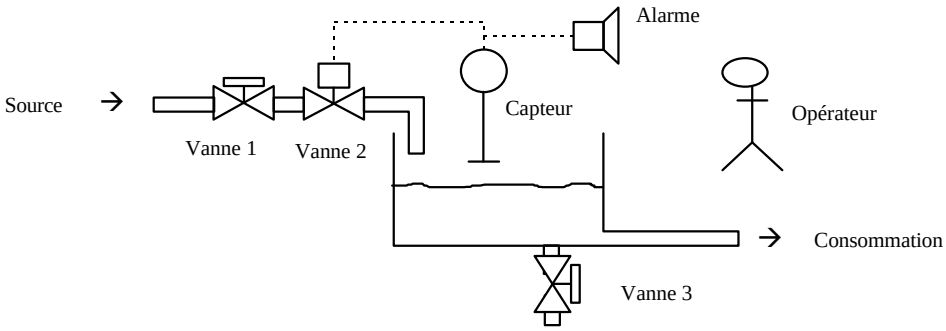


Figure 10.3 : Schéma de l'installation

L'événement redouté est le débordement du réservoir. En temps ordinaire, la quantité d'eau diminue en fonction de la consommation et augmente en fonction du débit de la source.
Si la consommation s'arrête, le niveau augmente jusqu'à ce que le capteur de niveau coupe automatiquement l'alimentation provenant de la source en fermant la vanne 2 et prévient un opérateur à l'aide d'une alarme. Ce dernier doit, en cas de défaillance de la vanne, fermer manuellement la vanne 1. La vanne 3 d'évacuation est le dernier recours de l'opérateur.
Un système équivalent a déjà été présenté dans le paragraphe 3.2, comme exemple méthodologique de construction d'un arbre de défaillance.

L'allocation consiste ici en la minimisation du coût de l'installation sous une contrainte probabiliste de sûreté (non débordement de la cuve). Pour chaque composant, différentes solutions techniques ont été envisagées et sont décrites dans le Tableau 10-1.
De plus :

- Pour des raisons de maintenance, les vannes 1 et 3 doivent être du même type.
- La pose de la vanne 3 coûte en plus du coût d'acquisition 50 U. Les concepteurs du système se demandent si la suppression de cette vanne peut être envisagée.
- Toute la sécurité du système est dépendante du capteur de niveau. Il est donc envisagé de mettre un second capteur en redondance active ; les deux capteurs étant du même type.

Composant	N° du choix	Coût*	Probabilité
Vannes manuelles	1	40	1.10^{-3}
V1 et V3	2	80	8.10^{-4}
	3	120	2.10^{-4}
Vanne commandée	1	100	2.10^{-3}
V2	2	180	7.10^{-4}
	3	250	2.10^{-4}
Capteur	1	50	5.10^{-3}
	2	120	1.10^{-3}
	3	190	4.10^{-4}
	4	320	1.10^{-4}
Alarme	1	20	1.10^{-4}
	2	60	1.10^{-6}

* : L'unité de coût est arbitraire (U)

Tableau 10-1 : Fonction de coût des composants

L'arbre de défaillance du système considéré peut être le suivant :

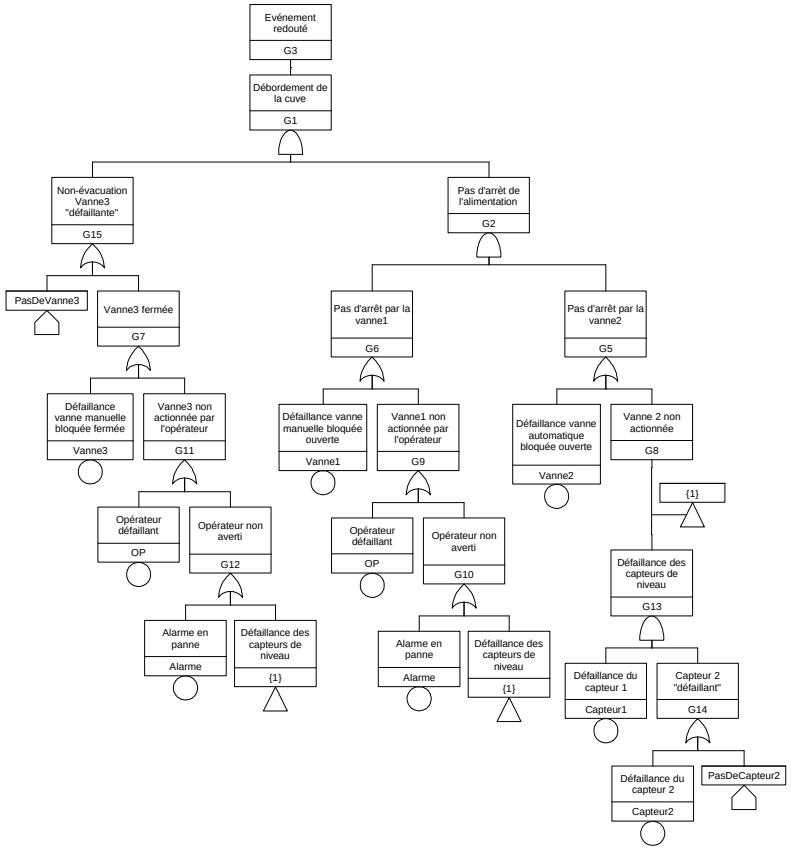


Figure 10.4 : Description de l'événement redouté, "Débordement de la cuve"

Deux événements de configuration (constantes nommées mises "à vrai" ou "à faux" au lancement d'un calcul) permettent de prendre en compte les choix d'architecture. `PasDeVanne3` doit être positionné "à vrai", pour ne pas prendre en compte la vanne 3. Il en est de même pour le capteur 2.

```
01    require aralia;        // précise l'utilisation du module externe aralia
02
03    model CmdNiveau {
04
05      aralia {{
06        /* Définition de la structure de l'arbre de défaillance  */
07        G1 := ( G2 & G15 ) ;
08        G2 := ( G5 & G6 ) ;
09        G3 := ( G1 ) ;
10        G5 := ( G8 | Vanne2 ) ;
11        G6 := ( G9 | Vanne1 ) ;
12        G7 := ( G11 | Vanne3 ) ;
13        G8 := ( G13 ) ;
14        G9 := ( G10 | OP ) ;
15        G10 := ( Alarme | G13 ) ;
16        G11 := ( G12 | OP ) ;
17        G12 := ( Alarme | G13 ) ;
18        G13 := ( Capteur1 & G14 ) ;
19        G14 := ( Capteur2 | PasDeCapteur2 ) ;
20        G15 := ( G7 | PasDeVanne3 ) ;
21        /* Définition des événements de configuration */
22        PasDeCapteur2 := 0;
23        PasDeVanne3 := 0;
24        /* Lois des événements de base non gérés par l'allocation */
25        law OP constant 0.005:
26      }};
27
28      /* Déclaration des variables de décision du problème */
29      decision int VanneManuelle  = [1,3];
30      decision int VanneCommandee = [1,3];
31      decision int Capteur        = [1,4];
32      decision int Alarme         = [1,2];
33      decision int Vanne3         = [0,1];
34      decision int Capteur2       = [0,1];
35
36      /* Définition des coûts unitaires des composants de base */
37      float CoutVanneManuelle  = switch VanneManuelle  {1:40, 2:80, 3:120} ;
38      float CoutVanneCommandee = switch VanneCommandee {1:100, 2:180, 3:250};
39      float CoutCapteur   = switch Capteur        {1:50, 2:120, 3:190, 4:320};
40      float CoutAlarme    = ite(Alarme == 1, 20, 60);
41
42      /* Définition du coût globales de l'installation */
43      float CoutTotal = CoutVanneManuelle             // Vanne 1
44                  + CoutVanneCommandee                // Vanne 2
45                  + Vanne3 * (CoutVanneManuelle + 50) // Vanne 3
46                  + CoutCapteur                       // Capteur 1
47                  + Capteur2 * CoutCapteur            // Capteur 2
48                  + CoutAlarme                        // Alarme
49                  ;
50
51      /* Affectation des lois de probabilité constante */
52      aralia("law", "Vanne1", "constant",
53        switch VanneManuelle {1:1e-3, 2:8e-4, 3:2e-4});
54      aralia("law", "Vanne2", "constant",
55        switch VanneCommandee {1:2e-3, 2:7e-4, 3:2e-4});
56      aralia("law", "Vanne3", "constant",
57        switch VanneManuelle {1:1e-3, 2:8e-4, 3:2e-4});
58      aralia("law", "Capteur1", "constant",
59        switch Capteur {1:5e-3, 2:1e-3, 3:4e-4, 4:1e-4});
60      aralia("law", "Capteur2", "constant",
61        switch Capteur {1:5e-3, 2:1e-3, 3:4e-4, 4:1e-4});
62      aralia("law", "Alarme", "constant",
63        switch Alarme {1:1e-4, 2:1e-6});
64      /* Affectation des événements de configuration */
65      aralia("constant", "PasDeVanne3", Vanne3 == 0);
66      aralia("constant", "PasDeCapteur2", Capteur2 == 0);
67
68      /* Calcul de la probabilité du sommet */
69      float ProbaSommet = aralia("Pr", "G1", 0);
70
```

```
71.     ./* Relation de dépendances */
72       dependenty ProbaSommet * -;
73       dependenty CoutTotal * +;
74     };
```

La variable de décision `Vanne3` (ligne 33) rend compte de la suppression possible de la vanne 3.
Lorsqu'elle est égale à zéro, nous considérons le cas où elle n'a pas été installée sur le système. Elle n'est pas prise en compte dans le coût du système (ligne 45) et l'événement de configuration de l'arbre de défaillance `PasDeVanne3` est mis "à vrai" (ligne 65). La branche décrivant la non-évacuation par la vanne 3 est alors inhibée (ligne 20 ou Figure 10.4).
A contrario (lorsque la variable de décision `Vanne3` est égale à un), l'événement de configuration `PasDeVanne3` est mis "à faux", il n'est pas pris en compte dans la logique de la porte `G15`. Le coût du système est alors fonction du coût de la vanne 3 augmenté du coût de l'installation de cette vanne.
Nous retrouvons la même logique de modélisation pour rendre compte du second capteur en redondance active.

Toutes les variables de décision rendent compte de choix permettant au système d'être de plus en plus fiable (indisponibilité décroissante), ce qui entraîne un coût de plus en plus élevé. Ces relations de dépendance sont précisées aux lignes 72 et 73.

10.4.2 LES PROBLEMES

Un problème décrit une question posée sur un modèle donné. Il peut y avoir plusieurs problèmes pour un modèle. Un problème est composé :

- De contraintes qui sont essentiellement des inégalités que devront vérifier toutes les solutions du problème. Typiquement, une contrainte de poids ou de volume peut être exprimée à l'aide d'une telle inégalité.
- D'une quantité à optimiser (à minimiser ou à maximiser).

Le problème fait référence à un modèle qui doit être déjà chargé en mémoire. Il est déclaré de la manière suivante :

```
problem CostSubjProb of CmdNiveau {
  optimize < costTot                    // Minimize costTot
  subject-to                            // Contrainte
    ProbaSommet < 5e-4;
};
```

La directive `optimize` est suivie d'un des symboles "<" (minimisation) ou ">" (maximisation), suivi de la quantité à optimiser.
La directive `subject-to` est suivie d'une liste d'inégalités séparées par des virgules. Chaque inégalité est de la forme <variable> <opérateur> <valeur> où <variable> est une variable intermédiaire, <opérateur> est un opérateur d'inégalité (>, <, >=, <=) et <valeur> une constante.

10.4.3 LES AFFECTATIONS

Une affectation sur un modèle correspond à une valuation totale ou partielle des variables de décision du modèle. Une affectation totale est valide pour un problème, si toutes les contraintes du problème sont satisfaites pour l'affectation.
Elles permettent de représenter les solutions d'un problème. Une affectation de départ est généralement fournie aux algorithmes comme première solution. Le résultat d'un algorithme de résolution est retourné à l'aide d'une affectation. Les affectations servent donc d'interface entre les algorithmes et l'utilisateur.
Elles font référence à un modèle qui doit être déjà chargé en mémoire.

```
assign Basic of CmdNiveau {
  VanneManuelle  = 1;
  VanneCommandee = 2;
  Capteur        = 3;
  Alarme         = 2;
  Vanne3         = 1;
  Capteur2       = 0;
};
```

Cet exemple déclare l'affectation `Basic` rattachée au modèle `CmdNiveau`. Il affecte à la variable de décision `Alarme` la valeur 2.

D'autres constructions permettent de définir rapidement des affectations typiques :

```
assign Min of CmdNiveau {}\<;     /* variable de décision a leur minimum */
assign Max of CmdNiveau {}\>;     /* variable de décision a leur maximum */
assign Moy of CmdNiveau {}\-;     /* variable de décision a une valeur moyenne */
assign Rnd of CmdNiveau {}\~;     /* variable de décision a une valeur aléatoire */
```

De nombreuses opérations sont possibles sur ces affectations. Il est ainsi possible de calculer la valeur de variables intermédiaires du modèle pour une affectation donnée.

```
Aloes > compute assign Min CoutTotal, ProbaSommet ;
        CoutTotal = 210
        ProbaSommet = 3.52e-5
```

10.5 ALGORITHMES

Tous les algorithmes proposés posent, comme préalable, que les variables de décision (Vd_i, pour i de 1 à n) du problème sont bornées ($min_i \leq Vd_i \leq max_i$), qu'il y a une grandeur à optimiser ($G(s)$) et que les k contraintes (autres que les bornes des variables de décision) sont données sous la forme d'une liste d'inégalités ($C_j(s)$ ($>$ ou $<$) B_j pour j de 1 à k). L'algorithme fonctionne d'autant mieux que la grandeur à optimiser et les variables intermédiaires des contraintes sont des fonctions monotones (croissantes ou décroissantes) des variables de décision du problème.

Un premier paragraphe présente différentes fonctions-objectif qui ont comme rôle de comparer deux solutions en tenant compte de la grandeur à optimiser, mais aussi des contraintes du problème. Cette fonction de comparaison est un élément de base de tout algorithme d'optimisation.

Les autres paragraphes de cette section présentent les différents algorithmes présents dans Aloès. L'algorithme de descente rapide cherche des solutions de manière stochastique, tandis que le Branch & Bound est un algorithme exhaustif. Ces deux premières méthodes s'appliquent exclusivement sur des variables de décision entières. *A contrario*, l'algorithme du Simplex ne traite que des problèmes ayant des variables de décision réelles.

Nous nous sommes volontairement limités à ces algorithmes. Ce choix s'explique principalement par le fait qu'il est illusoire de chercher à ajouter ou à améliorer un algorithme s'il n'existe pas de contexte d'étude et donc de besoins réels pour les industriels. Nous avons donc choisi des algorithmes élémentaires. Libre à nous d'en ajouter d'autres dans l'avenir si les résultats de nos expérimentations ne nous satisfont pas.

10.5.1 FONCTION OBJECTIF

Toutes les solutions d'un problème d'optimisation n'ayant pas les mêmes qualités, il convient d'avoir un moyen de sélectionner la "meilleure". L'évaluation de la qualité d'une solution peut se faire à l'aide d'une fonction M appelée fonction-objectif (ou mesure) telle que, si s_1 et s_2 sont deux solutions potentielles d'un problème et si s_1 est de meilleure qualité que s_2, on doit alors avoir $M(s_1) < M(s_2)$. Une fonction-objectif définit donc ce que l'on doit optimiser.

Une affectation donnée est une solution à un problème lorsqu'elle satisfait toutes les contraintes du problème. Tous les algorithmes travaillent en comparant, non pas des solutions, mais des affectations. La difficulté intervient lorsque l'on doit comparer deux affectations qui ne satisfont pas les contraintes. La fonction-objectif doit alors prendre en considération, en plus de la grandeur à optimiser, les contraintes du problème. En effet, les contraintes, diminuant l'espace des solutions du problème, doivent être pénalisantes pour la fonction-objectif lorsqu'une solution considérée ne les respecte pas.

Dans le cas d'une optimisation sans contrainte, la fonction-objectif est égale à la grandeur à optimiser. Dans le cas contraire, il est d'usage d'ajouter ou de soustraire (suivant le type d'optimisation) à la grandeur à optimiser une fonction positive dite fonction de pénalisation (ou de pondération) qui dépend des violations des contraintes.
De nombreuses fonctions de pénalisation existent dans la littérature. Nous en présentons trois ci-dessous :

- Une des manières de prendre en considération les contraintes est d'ajouter une pénalité constante lorsqu'une contrainte n'est pas satisfaite. La pénalité doit être suffisamment importante par rapport à la grandeur à optimiser, car, dans le cas contraire, une solution optimisant fortement G, mais ne respectant pas les contraintes, pourrait être renvoyée par l'algorithme.

  $$M(s) = G(s) \pm (k \times W)$$

  avec k : nombre de contraintes non satisfaites [10-4]

  W : pénalité constante

  Lors de la comparaison de deux solutions ne satisfaisant pas les contraintes, la fonction objectif ne pourra pas les différencier correctement. *A priori*, elle avantagera même plutôt la solution optimisant la grandeur G qui doit, dans la majorité des cas, être celle qui satisfait le moins les contraintes (La grandeur G et les contraintes étant le plus souvent antinomiques).
  Lorsque l'algorithme améliore une solution courante de manière itérative, une pénalité constante ne peut être utilisée que si le point de départ de l'algorithme satisfait les contraintes. Cette méthode de pénalisation consiste à approcher la solution par l'intérieur du domaine des solutions admissibles.

- A l'opposé la fonction [10-5] permet d'approcher l'optimum par l'extérieur. Cette fonction de pénalisation offre l'avantage de pouvoir comparer deux solutions quelconques avec l'idée que moins une contrainte est respectée, plus la pénalisation est forte.

  $$M(s) = G(s) \pm \sum_{j=1}^{k} \left(p_j(s) \times W_j \times \left(C_j(s) - B_j \right)^2 \right)$$

  avec $p_j(s) = 0$ si la contrainte j est respectée [10-5]

  $p_j(s) = 1$ si la contrainte j ne l'est pas

  et W_j Coefficient de pénalité de la contrainte j (strictement positif)

  En revanche, même si nous fixons des W_j suffisamment grands, nous ne sommes pas à l'abri de trouver une solution ne satisfaisant pas complètement les contraintes du problème. Lorsque $C_j(s)$ tend vers B_j mais ne satisfait pas la solution, la pénalité associée tend vers zéro.

- La fonction suivante prend en compte les contraintes sous la forme d'une pénalité d'un poids minimum W_j.

  $$M(s) = G(s) \pm \sum_{j=1}^{k} \left(p_j(s) \times W_j \times \left(1 + \frac{\left| C_j(s) - B_j \right|}{B_j} \right) \right)$$

  avec $p_j(s) = 0$ si la contrainte j est respectée [10-6]

  $p_j(s) = 1$ si la contrainte j ne l'est pas

  et W_j Coefficient de pénalité de la contrainte j (strictement positif)

  La distance relative entre une contrainte et sa limite d'acceptabilité est prise en compte, ce qui permet de trier deux solutions ne respectant pas une même contrainte en avantageant celle qui satisfait le plus la contrainte. Lorsque la contrainte n'est pas respectée, la pénalité minimum est égale à W_j. Il convient de fixer W_j suffisamment grand comparativement à $G(s)$ et de telle manière que le coefficient de pénalité d'une contrainte rende compte de la difficulté à satisfaire cette dernière.

Ces trois fonctions sont disponibles au sein de Aloès sous la forme de paramètres des différents algorithmes. Nous verrons dans la partie expérimentation une utilisation en deux passes de l'algorithme du Simplex. La

première passe utilise la dernière fonction de pénalisation, afin d'obtenir une affectation valide (qui satisfait les contraintes). Puis, à partir de cette affectation, une seconde passe permet de s'approcher au plus près de la solution à l'aide de la première fonction de pénalisation.

10.5.2 DESCENTE RAPIDE

Les méthodes de réparation locale [Ree95, AL97] (ou méthodes stochastiques) partent d'une configuration initiale et appliquent des transformations élémentaires successives à la solution courante tant qu'un critère d'arrêt n'est pas vérifié.
La mise en œuvre des méthodes stochastiques nous amène à choisir une configuration initiale, une transformation élémentaire (un voisinage) , une fonction-objectif et une condition d'arrêt.
Une transformation élémentaire consiste à changer une caractéristique de la solution courante ; par exemple, la modification d'une variable de décision entière de + ou - 1.
Une condition d'arrêt peut être définie par l'utilisateur comme étant le nombre maximum de cycles de l'algorithme.
La configuration initiale est définie à l'aide d'une affectation des variables de décision. Cette affectation peut être créée en fixant les variables de décision à leur borne minimale ou maximale ou à une valeur aléatoire comprise dans les domaines de définition.

La Figure 10.5 montre l'existence de minima locaux. Ceci impose l'utilisation de méthodes stochastiques efficaces pour éviter de rester bloqué aux alentours de ces minima en dirigeant la recherche vers la solution optimale.

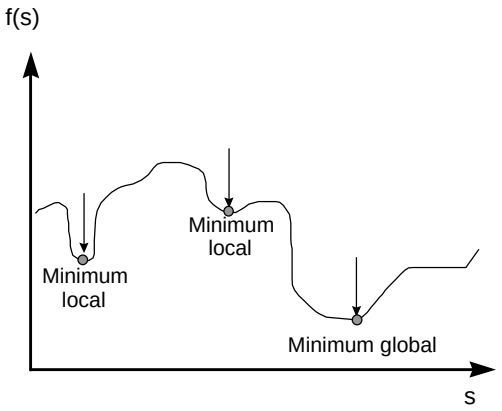


Figure 10.5 : Fonction à optimiser en fonction de s (solution)

De nombreuses méthodes ont été proposées. Parmi celles-ci citons à nouveau :

- Les descentes rapides qui ne considèrent une transformation que si elle améliore la solution courante.
- Le recuit simulé, basé sur l'équilibre énergétique lors de la cristallisation des métaux.
- La recherche Tabou, qui introduit une notion de mémoire dans la stratégie d'exploration des solutions.
- Les algorithmes génétiques, qui considèrent les solutions comme des individus, dont seuls les mieux adaptés sont aptes à se reproduire.
- ...

Une variante de la méthode de descente rapide a été mise en œuvre dans Aloès. Ce premier choix repose sur la facilité de conceptualisation et d'implémentation de cette méthode.

Le principe de cette méthode est relativement simple. A partir d'une solution courante s, une solution *s'* est recherchée par transformations élémentaires. Si, dans le cas d'une minimisation, $M(s') < M(s)$, notre solution courante devient *s'*, sinon *s* reste la solution courante. Lorsqu'aucune transformation élémentaire appliquée à la solution courante n'améliore cette dernière, l'algorithme a atteint un minimum local. Il n'est donc plus possible de poursuivre.

De nombreuses améliorations à cette méthode peuvent être envisagées. Nous avons considéré dans un premier temps les améliorations décrites par l'algorithme de la Figure 10.6.

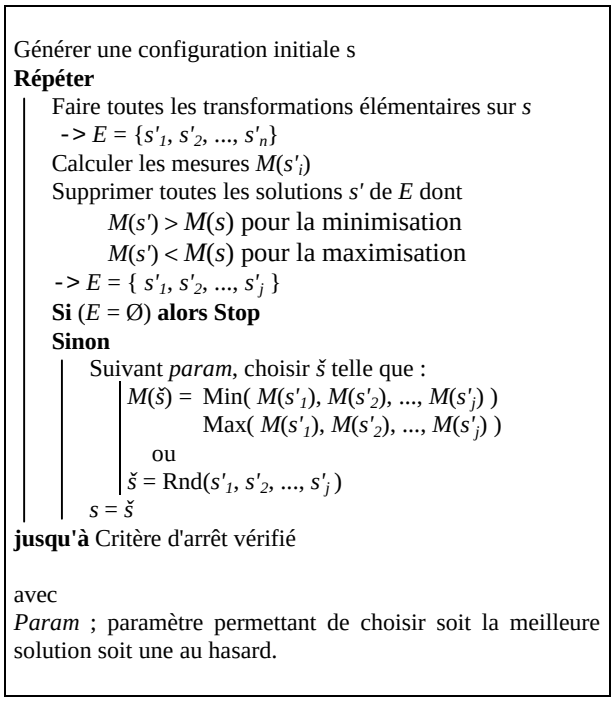


Figure 10.6 : Algorithme de descente rapide

L'intérêt de ces améliorations est de regarder l'ensemble des solutions voisines de *s* et d'en choisir une comme successeur.
Ce choix se fait suivant le paramètre *param*, qui est réévalué à chaque cycle. Cela permet de choisir, dans un cycle, comme successeur la meilleure solution, puis au cycle suivant une au hasard parmi celles qui améliorent la solution. Il s'agit donc d'un algorithme à cheminement aléatoire (randomisé).
Le hasard jouant un rôle important dans la recherche du minimum, la possibilité de jouer *m* configurations initiales permet à l'algorithme de partir de situations différentes et de faire des choix différents lors de son déroulement. C'est donc aussi un algorithme itéré.

10.5.3 BRANCH & BOUND

L'algorithme du Branch & Bound (Evaluation et Séparation) est une amélioration d'un algorithme d'exploration exhaustive de l'espace des solutions.
Cette exploration se fait par un parcours dans l'arbre de recherche (Figure 10.7). A chaque nœud de l'arbre, on considère l'ensemble des valeurs possibles pour la variable de décision considérée. Il convient donc de se limiter aux variables de décision discrètes. L'exploration se fait en profondeur à gauche.

Une exploration exhaustive ne permet pas de traiter des problèmes de taille industrielle (N>10). En effet, le temps de traitement croît de manière exponentielle avec la taille du problème à résoudre.

Afin de réduire l'explosion combinatoire du problème, la méthode du Branch & Bound évalue la branche en cours pour déterminer s'il existe une solution potentielle dans cette partie de l'arbre qui soit meilleure que celles trouvées précédemment. Si ce n'est pas le cas, l'exploration de la branche courante est abandonnée.
Soit G, la grandeur à optimiser. Lors de l'évaluation de la branche $VD_iCh_2VD_{i+1}$, les valeurs des variables de décision VD_k (k de 1 à i) sont déjà fixées. Un opérateur de minimisation (respectivement maximisation) sur G peut déterminer, lorsque cela est possible, les valeurs des variables de décision VDj (j de i+1 à n) qui

minimisent (respectivement maximisent) la grandeur G. Si la borne inférieure (resp. supérieure) est supérieure (resp. inférieure) à la valeur courante de G, il n'est pas nécessaire d'explorer la suite de cette branche.
Il est possible d'utiliser des techniques similaires sur les contraintes en considérant chacune d'elles comme une grandeur $C_j(s)$ à optimiser avec B_j comme valeur courante de $C_j(s)$.

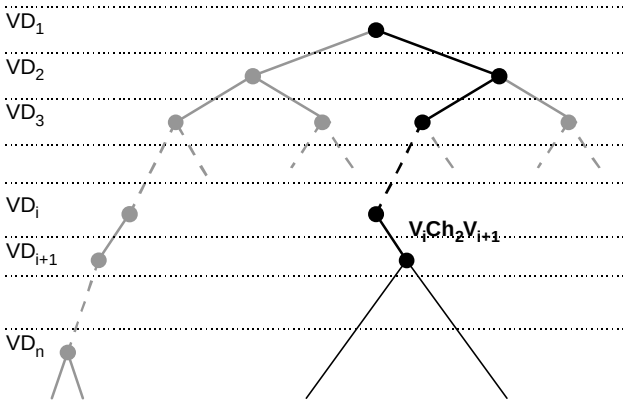


Figure 10.7 : Arbre de recherche

L'opérateur de minimisation est facilement réalisable à partir des relations de dépendance entre les grandeurs à optimiser et les variables de décision libres, c'est-à-dire celles qui n'ont pas encore été fixées. Si la grandeur est non monotone en fonction d'une des variables libres, l'opérateur de minimisation ne peut pas donner de résultat, en tout cas pas suffisamment rapidement pour pouvoir être considéré. Si la grandeur est monotone en fonction de toutes les variables libres, il suffit de fixer la valeur de chaque variable à une des bornes de son intervalle de variation en fonction de la relation existant entre la grandeur à optimiser et la variable de décision (croissante ou décroissante) et du type d'optimisation (recherche d'un minimum ou d'un maximum).
Il en est de même pour l'opérateur de maximisation.

L'ordre du parcours de l'arbre de recherche est déterminant dans la rapidité de l'algorithme. Il peut être choisi soit une fois pour toute en début de traitement (nous parlons alors d'une heuristique statique) soit réévaluer à chaque pas de l'algorithme; la prochaine variable de décision à traiter dépend des variables qui ont été fixées au préalable. Dans tous les cas ces heuristiques ont pour but de faciliter les techniques de séparation. Il s'agit donc de déterminer quelles sont les variables de décision à explorer en premier lieu dans l'arbre de recherche afin de séparer au plutôt les branches ne pouvant pas fournir de solutions satisfaisantes.
Nous avons vu dans les paragraphes précédents que les techniques de séparation considérées ne fonctionnaient pas pour une grandeur à optimiser lorsque cette dernière n'était pas monotone en fonction d'une variable libre. Les variables non monotones pour une ou plusieurs grandeurs à optimiser seront donc à traiter rapidement, car elles empêchent l'application des techniques de séparation.
Les grandeurs à optimiser sont plus ou moins sujettes aux variations des variables de décision. Il conviendra de traiter en premier lieu la variable de décision qui engendrera la variation la plus importante des grandeurs à optimiser. Fixer en premier ce type de variables de décision, permet de tronquer au plus tôt les branches qui ne peuvent contenir de solutions satisfaisantes (ne répondant pas aux contraintes du problème ou dont la borne supérieure ou inférieure de la grandeur à optimiser est inférieure ou supérieure à la meilleure valeur déjà trouvée).

Une fois l'ordre des variables déterminé, il est encore possible d'améliorer la rapidité de l'algorithme en choisissant pour une variable donnée l'ordre d'affectation de ses valeurs (compris entre sa borne minimum et maximum). L'heuristique, que nous avons retenue, vise systématiquement à choisir un ordre qui permet d'aller au plus vite vers l'optimum sans considérer les contraintes du problème. Soit *G*, une grandeur à minimiser, monotone décroissante en fonction d'une variable de décision vd_i bornée sur le domaine $[min_i, max_i]$. L'ordre d'affectation de la variable vd_i se fera de max_i à min_i. De cette manière, nous explorons en premier lieu les branches où une solution est potentiellement meilleure que dans les autres branches.

Lors d'une utilisation d'un outil comme Aloès, nous supposons bien entendu qu'une solution a déjà été trouvée à l'aide d'un autre algorithme, tel qu'une descente rapide. Nous souhaitons donc savoir s'il existe une meilleure solution à l'aide d'un algorithme exhaustif du type Branch&Bound. Grâce à cette première solution nous disposons d'une borne d'assez bonne qualité pour la grandeur à optimiser.

10.5.4 SIMPLEX

L'algorithme du Simplex de Nelder et Mead [NM65], ou du polyèdre flexible, est une méthode d'optimisation multi-directionnelle robuste. Un "Simplex" est une figure géométrique constituée de (N+1) sommets, en considérant que l'on travaille dans un espace à N dimensions. La méthode débute, non pas avec un point unique, mais avec un Simplex initial; ensuite au moyen d'une séquence de transformations géométriques élémentaires (réflexion, contraction, extension, ...), le Simplex se distord de façon à s'adapter au relief de la fonction et finalement vient se contracter vers un optimum.
Un nouveau Simplex est généré à partir de l'ancien par homothétie d'un sommet par rapport au centre de gravité des sommets restants. Le nouveau Simplex est alors formé en remplaçant le sommet projeté par celui nouvellement calculé. Le paramètre d'homothétie θ prend différentes valeurs fixées par l'utilisateur en fonction des phases de l'algorithme (réflexion $\theta<0$, contraction $0<\theta<1$, extension $\theta>1$, ...).
Le cycle s'arrête lorsque la distance entre le meilleur point du Simplex et le moins bon est inférieure à une limite fixée. Le Simplex est alors contracté autour d'un optimum. Afin d'éviter un arrêt prématuré de la méthode, un Simplex est créé à partir du meilleur point du Simplex précédent. Le processus est ainsi répété un certain nombre de fois.

L'algorithme mis en œuvre dans Aloès est décrit de manière succincte à l'aide du pseudo-code suivant :

01 Soit $S = \{P_0, P_1, ..., P_N\}$ le Simplex génère à partir d'un point P_0.
02 Calculer la fonction objectif pour chacun des points du Simplex : $M(P_0)$, ..., $M(P_N)$
03 Tant que (critère d'arrêt n'est pas vérifié) :
04 Soit P_H la moins bonne solution du Simplex : $M(P_H) = \text{Max}(M(P \in S))$
05 Soit P_{HP} la moins bonne solution du Simplex en dehors de P_H: $M(P_{HP}) = \text{Max}(M(P \in S/\{P_H\}))$
06 Soit P_B la meilleure solution du Simplex : $M(P_B) = \text{Min}(M(P \in S))$
07 Soit C le centre de gravité du Simplex sans P_H : $S / \{P_H\}$
08 Calculer P' = Homothétie(P_H, C, α) avec $\alpha<0$ ***(Réflexion)***
09 Si ($M(P') < M(P_H)$)
10 $P_H \leftarrow P'$
11 Si ($M(P_H) < M(P_B)$)
12 Calculer P' = Homothétie(P_H, C, γ) avec $\gamma>1$ ***(Extension)***
13 Si ($M(P') < M(P_B)$) $P_H \leftarrow P'$
14 **Aller à Suite**
15 Si ($M(P_H) < M(P_{HP})$) **Aller à Suite**
16 Calculer P' = Homothétie(P_H, C, β) avec $0<\beta<1$ ***(Contraction)***
17 Si ($M(P') < M(P_H)$) $P_H \leftarrow P'$
18 Sinon ***(Rétraction vers la meilleure solution)***
19 $P_i \leftarrow$ Homothétie(P_i, P_B, μ) $\forall\ i \neq B$ avec $\mu > 0$
20 **Suite**

La fonction Homothétie(P, C, θ) renvoie le point homothétique à P par rapport au centre C et de paramètre θ.
Le critère d'arrêt du cycle a été défini de la manière suivante :

$$\frac{M(P_H) - M(P_B)}{\frac{1}{2}\left(M(P_H) + M(P_B)\right)} < e \qquad [10\text{-}7]$$

Lorsque le critère d'arrêt est vérifié le cycle s'arrête, un nouveau Simplex S est alors généré à partir du point P_B et un nouveau cycle commence. L'algorithme prend fin lorsque la solution de fin de cycle n'a pas été améliorée pendant un nombre de cycles préalablement défini.

Le schéma ci-contre met en œuvre de manière graphique l'algorithme et montre l'intérêt de ne pas se limiter uniquement à une transformation géométrique de type Réflexion.
Pour cet exemple, nous cherchons à minimiser la distance entre un objectif et le point courant défini par deux variables de décision. Nous travaillons dans un espace à 2 dimensions, notre Simplex est donc formé de trois points.
Le premier algorithme (ne prenant en compte que la réflexion : $\alpha = -1$, $\beta = \gamma = \mu = 0$) trouve une solution après 13 itérations.

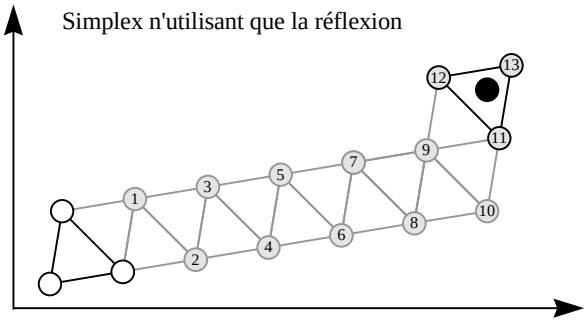


Figure 10.8 : Simplex utilisant uniquement la réflexion

Le second ($\alpha = -1$, $\beta = 0.5$, $\gamma = 2$, $\mu = 0$) donne une solution de meilleure qualité après 7 itérations. La lettre située à côté du numéro de l'itération correspond aux différentes phases de l'algorithme (a : réflexion, b : extension, c : contraction).
Le nombre d'évaluations de la fonction objectif est globalement identique (13 dans le premier cas, 14 (7x2) dans le second).

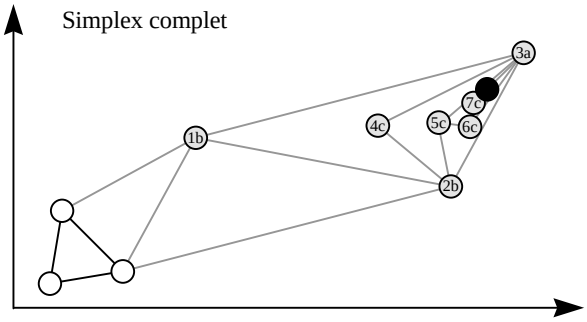


Figure 10.9 : Simplex complet

10.6 BIBLIOGRAPHIE

[AL97] E.H.L. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997. ISBN 0-471-94822-5.

[BB97a] M. Bouissou and E. Bourgade. *Evaluation et allocation d'indisponibilité à la conception des tranches nucléaires : Méthodes et Outils.* Revue Française de Mécanique n°1997-2, 1997, pp105-111

[BB97b] M. Bouissou and E. Bourgade. *Unavailability Evaluation and Allocation at the Design Stage for Electric Power Plant : Methods and Tools.* Proceeding Annual Reliability and Maintenability Symposium IEEE, 1997, pp91-99

[DMRT98a] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Sherloc : a Desk Calculator for Availability Allocation in Fault Trees* Proceedings of the European Safety and Reliability Association Conference, ESREL'98.

[DMRT98b] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Conception Optimization Based on the Definition of Availability Allocation Taking into Account Cost Constraint.* Proceedings of the International Conference of Probabilistic Safety Assessment and Management, PSAM'4, volume 2, pages 1591-1596, New-York, 1998. Springer Verlag.

[DMRT98c] H. Desille, F. Meunier, A. Rauzy et P. Thomas. *Sherloc : outil d'allocation d'indisponibilité pour les arbres de défaillance.* Actes du 11ième colloque national de Fiabilité et Maintenabilité, λμ11, 1998.

[MRT97] F. Meunier, A. Rauzy, and P. Thomas. *Utilisation de la méthode du gradient descendant pour l'allocation de fiabilité.* Actes de la 3ième Conférence Nationale sur la Résolution Pratique de Problèmes NP-Complets, pp109-112, 1997.

[NM65] J.A. Nelder, R. Mead. *A simplex method for function minimisation*. Computer Journal, 1965

[PK00] V. Prasad and W. Kuo. *Reliability Optimization of Coherent Systems.* IEEE Transactions on Reliability Vol. 49 No. 3, September 2000, pp323-330

[Ree95] C.R. Reeves, editor. *Modern Heuristics Techniques for Combinatorial Problems*. MacGraw Hill, 1995. ISBN 0-07-709239-2.

11. EXPERIMENTATION

De nombreux exemples issus de la littérature ont été traités à l'aide d'Aloès afin de valider le pouvoir d'expression du langage et de comparer la précision des résultats.

Dans ce chapitre, nous présentons les résultats obtenus sur un banc d'essais constitué d'une vingtaine de réseaux de fiabilité de la littérature. Nous avons plus particulièrement étudié les problèmes suivants :

- allocation de redondance active en prenant ou non en compte les défaillances de cause commune ;
- allocation de maintenance : optimisation d'une politique de remplacement périodique ;
- allocation en conception préliminaire dont les sous-systèmes sont caractérisés par une fonction d'effort de Truelove.

Pour chacun de ces modèles, nous nous sommes posés les questions suivantes :

- Combien cela coûte d'améliorer la fiabilité du système d'un facteur donné ?
- Comment doit évoluer la fiabilité du système, si l'on souhaite faire une économie d'un certain pourcentage ?

11.1 RESEAUX DE FIABILITE ETUDIES

Les expérimentations présentées dans la littérature considèrent des systèmes en général triviaux (k sur n, série - parallèle, ...). Dans la grande majorité des cas, il s'agit de systèmes booléens dont l'indisponibilité peut être calculée à l'aide d'une formule littérale. L'avantage est qu'il n'est alors pas nécessaire de coupler la méthode à des outils d'évaluation du modèle probabiliste. Le désavantage principal est que la taille des problèmes considérés peut difficilement dépasser quelques dizaines de composants. De plus, un ingénieur chargé de l'étude d'optimisation devra déterminer la formule littérale du système considéré ce qui n'est pas nécessairement facile. Par ailleurs, s'il est possible de mettre en place des heuristiques spécifiques adaptées à des systèmes de ce type, elles ne seront pas exploitables pour des systèmes de nature différente.

Les réseaux de fiabilité en revanche offrent de nombreux avantages d'un point de vue méthodologique :

- Ils peuvent être complexes et, en particulier, de taille suffisante pour que l'on ne puisse plus rechercher "à la main" de bonnes heuristiques.
- Ce sont forcément des systèmes cohérents : plus un composant est fiable, plus le système l'est.
- Leur représentation graphique a le même avantage que les diagrammes de fiabilité. Elle s'appuie sur une vue fonctionnelle du système, qui est plus familière aux praticiens.

Il y a dans la littérature peu d'articles qui traitent de l'optimisation des réseaux de fiabilité.
Misra et Sharma [MS91] étudient, entre autre, deux réseaux de fiabilité (un réseau de type "bridge" à 5 éléments et un réseau de type ARPA de 5 nœuds et 7 arêtes). Ils considèrent une optimisation de redondance sans prendre en compte les défaillances de cause commune. Plusieurs jeux de paramètres (fiabilité et coût des composants de base, ainsi que seuil maximum pour le coût total) sont évalués. Dans le même article, ils étudient l'architecture d'un réseau de communication. Chaque arête du réseau est optionnelle (variante d'architecture), coûte un certain prix, tombe en panne et peut être réparée en fonction respectivement d'un taux de défaillance et d'un taux de réparation. L'optimisation cherche à maximiser la disponibilité de l'installation sous contrainte de coût.
Il semble qu'aucune étude portant sur un grand nombre de réseaux de fiabilité et débouchant sur des propositions de modélisation variées n'ait été publiée. Nous avons entrepris une telle étude dans le but de tester essentiellement la robustesse des algorithmes mis en œuvre.

Soh et Rai ont répertorié dans [SR93] (repris aussi dans [KLY99]) 19 réseaux de fiabilité publiés dans la littérature. Ils sont réunis à la Figure 11.1. Ces réseaux sont numérotés par ordre croissant de complexité.

Les liens sont bidirectionnels, sauf indication contraire. Les sommets blancs sont les sommets-sources, les noirs, les sommets-destinations et les gris, les autres sommets du réseau. Ces réseaux ont été redessinés par rapport à [SR93] afin de mettre en évidence leur aspect régulier. Cela permet de se rendre compte rapidement

que les réseaux SR93-9, SR93-10 et SR93-15 sont très proches. Le réseau SR93-16, et dans une moindre mesure les réseaux SR93-7 et SR93-13, semblent beaucoup plus réguliers sur ce schéma que dans la référence d'origine.

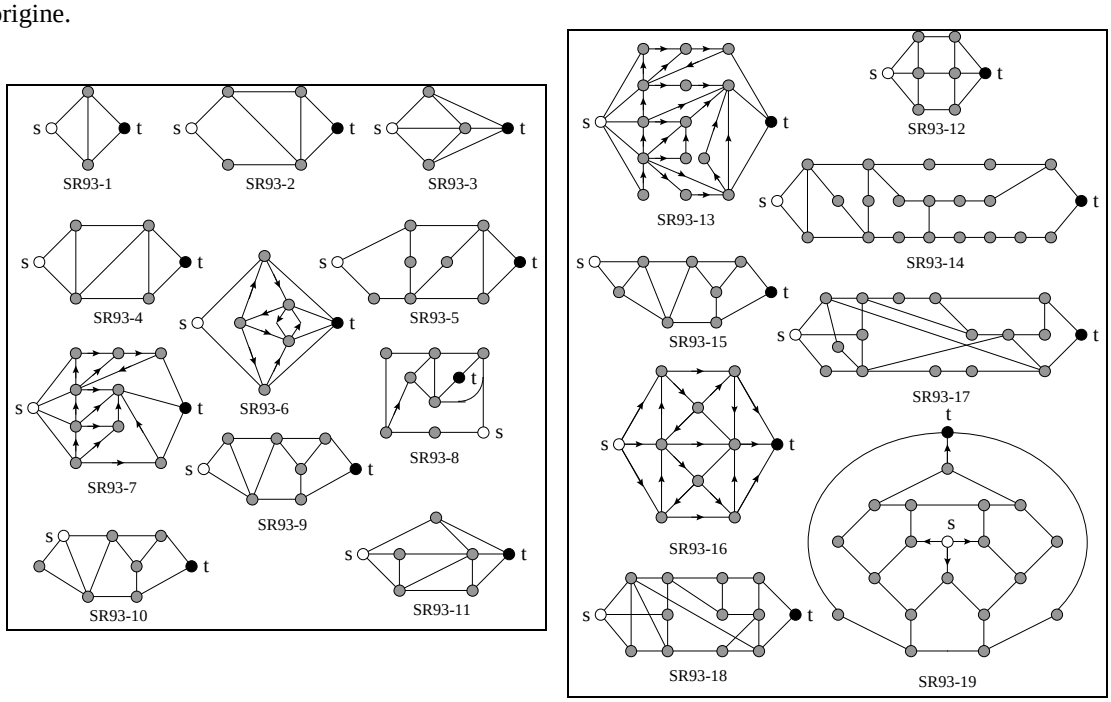


Figure 11.1 : Réseaux de fiabilité utilisés comme "benchmark"

Le Tableau 11-1 représente le nombre de nœuds, d'arêtes et de coupes minimales (ainsi que leur distribution) pour chaque réseau de fiabilité.

Réseau de fiabilité	Nbr de nœuds	Nbr d'arêtes	Nombre de coupes minimales par ordre							
			Total	#2	#3	#4	#5	#6	#7	>#7
SR93-1	4	5	4	2	2	-	-	-	-	-
SR93-2	6	8	8	-	2	4	2	-	-	-
SR93-3	5	8	9	3	3	3	-	-	-	-
SR93-4	6	9	9	2	3	2	2	-	-	-
SR93-5	9	12	28	3	7	6	12	-	-	-
SR93-6	7	14	23	-	1	3	10	8	1	-
SR93-7	11	21	110	-	6	8	17	31	30	18
SR93-8	9	13	24	3	2	7	4	8	-	-
SR93-9	8	12	19	2	5	5	4	3	-	-
SR93-10	8	12	20	1	6	4	5	4	-	-
SR93-11	7	12	20	-	2	5	5	5	3	-
SR93-12	8	13	29	-	3	8	10	4	4	-
SR93-13	16	30	364	-	2	9	45	58	58	192
SR93-14	21	26	528	6	66	72	96	108	180	-
SR93-15	9	14	25	2	6	5	4	5	3	-
SR93-16	10	21	78	-	2	4	14	24	27	7
SR93-17	16	25	844	1	4	9	38	72	134	586
SR93-18	13	22	219	1	2	7	17	24	29	139
SR93-19	20	28	3481	-	6	15	66	284	734	2376

Tableau 11-1 : Coupes minimales des réseaux de fiabilité du "benchmark"

11.2 PROTOCOLE EXPERIMENTAL

Nous considérons dans la suite de ce chapitre que tous les nœuds sont sûrs, c'est-à-dire qu'ils ne connaissent pas de défaillance. Seules les arêtes des réseaux peuvent défaillir.
De plus, toutes les arêtes seront soumises aux mêmes règles de définition de leur probabilité et de leur coût. Cette restriction appauvrit évidemment le problème, car, dans la réalité, les concepteurs de ces systèmes traiteront chaque arête en fonction de sa criticité. En revanche, cette restriction n'a que peu d'influence sur l'objectif de ces expérimentations qui est, rappelons-le, de tester la robustesse des algorithmes proposés. Les réseaux étant très disparates, le résultat de l'optimisation prend donc fortement en compte leur architecture.

Le langage Aloès (en particulier le couplage avec des modules de calcul extérieurs) permet d'étudier relativement aisément une classe de problème. Dans notre cas, nous avons réalisé des générateurs de modèles prenant comme données d'entrée un réseau de fiabilité et des paramètres d'étude (comme le coût et l'indisponibilité de référence des arêtes) et fournissant en sortie un modèle, une affectation de référence et des problèmes à étudier au format Aloès.

Nous verrons par la suite que les trois modélisations considérées permettent de définir, pour une instance du problème, un modèle et une affectation de référence. Ce modèle définit une grandeur fiabiliste (*GFiab*) et un coût global (*Cost*) qui sont fonctions de ses variables de décision. Il est donc possible, à partir de l'affectation de référence, de calculer la valeur de référence de ses grandeurs : $GFiab_{Ref}$ et $Cost_{Ref}$.
A partir d'un système donné, de sa grandeur fiabiliste et de son coût de référence, le décideur peut se poser les quatre questions suivantes :

1. Est-il possible d'augmenter la rentabilité du système sans le dégrader ?
2. Combien coûte une amélioration du système d'un facteur *CoefF* ?
3. Quelle est la meilleure configuration possible du système pour un coût de référence donné ?
4. Comment évolue le système, si j'en diminue le coût d'un facteur *CoefC* ?

Les questions 1 et 3 considèrent que le système actuel n'est pas optimisé. Elles recherchent donc une meilleure configuration avec une contrainte sur la fiabilité dans le cas 1, ou sur le coût dans le cas 3.
Pour répondre à chacune de ces questions, il faut résoudre différents problèmes d'optimisation :

1. Minimiser *Coût* sachant que $GFiab < GFiab_{Réf}$.
2. Minimiser *Coût* sachant que $GFiab < GFiab_{Réf} \times CoefQ$.
3. Minimiser *GFiab* sachant que $Cost < Cost_{Réf}$.
4. Minimiser *GFiab* sachant que $Cost\ t < Cost_{Réf} \times CoefC$.

L'expérimentation a pour objectif de montrer qu'il est possible de répondre à ces quatre questions pour des systèmes de tailles très différentes.
Si la diversité des systèmes a permis d'étudier la robustesse des algorithmes, il ne nous sera pas possible dans ce manuscrit de présenter tous les résultats pour l'ensemble des réseaux de fiabilité. Par contre, lorsque nous considérerons que des détails sont nécessaires pour étayer l'argumentation, nous utiliserons le réseau de fiabilité SR93-9 qui est représentatif des autres réseaux étudiés.

11.3 ALLOCATION DE REDONDANCE

Kuo et Prasad dans [KP00] répertorient 83 méthodes d'optimisation de la fiabilité depuis 1977. Parmi celles-ci 54 s'intéressent particulièrement à l'allocation de redondance et seules 2 d'entre elles prennent en compte les défaillances de cause commune.

11.3.1 PRINCIPE DE L'ALLOCATION DE REDONDANCE

L'idée de base est de considérer chaque arête du réseau comme un équipement composé de n composants en parallèle (redondance active) ; *n* étant un entier borné entre *min* et *max*.
Le coût d'une arête est déterminé par le coût unitaire du composant (C_{el}): $C_s = n \times C_{el}$. Nous négligeons les éventuelles économies d'échelle lors de l'achat de *n* composants similaires.
De la même manière, et en négligeant totalement la mise en œuvre de la redondance et les défaillances de cause commune entre les composants, la fiabilité de l'arête est donnée par la formule $R_s = 1-(1-R_{el})^n$, où R_{el} est

la fiabilité du composant. La probabilité de défaillance d'une arête Q_s (défiabilité dans le cas de système non réparable) est égale à $(Q_{el})^n$.
Le coût du système est défini comme étant la somme des coûts des arêtes. Les paramètres de l'étude (C_{el}, Q_{el}, *min* et *max*) sont les mêmes pour toutes les arêtes du réseau.

11.3.2 PRISE EN COMPTE DES DEFAILLANCES DE CAUSE COMMUNE

Les défaillances de cause commune (DCC) correspondent, en général, aux erreurs de conception, de fabrication, ou de maintenance, ..., réalisées pour des composants identiques d'un même système. Des composants identiques en redondance peuvent être soumis à des défaillances de cause commune. Typiquement, une équipe de maintenance peut commettre la même erreur sur plusieurs pompes du même type fonctionnant à la sollicitation. Toutes ces pompes poseront le même problème lors de leur sollicitation de démarrage.
D'un point de vue plus formel, une DCC est un événement pouvant entraîner la défaillance de plusieurs composants élémentaires du système. Une inondation dans un local peut entraîner la défaillance de tous les composants situés dans ce local.
Il existe des manières qualitatives pour mettre en évidence les risques de DCC. Elles consistent principalement à identifier, au niveau du système, les coupes contenant plusieurs composants localisés au même endroit, provenant du même fabricant ou sujet aux mêmes types d'agression externes.
Les méthodes quantitatives ne fonctionnent en général que pour les DCC de composants identiques. La méthode la plus facile à appréhender et la plus simple à mettre en œuvre est la méthode du β-facteur. Elle consiste à considérer que pour un ensemble de *n* composants identiques, chaque composant peut soit être le seul en panne, soit être en panne en conjonction avec les autres composants de l'ensemble.
Soit un groupe de DCC *Gr* composé des défaillances élémentaires E_1, E_2, E_3.
Le principe de base est de redéfinir E_i de la manière suivante :

$$E_i = E_{iS} \vee Gr \qquad [11\text{-}1]$$

où E_{iS} est la défaillance du composant *i* seul
et *Gr* est la défaillance de tous les composants du groupe *Gr*.

Soit Q_{el} l'indisponibilité de ces composants et β, un réel compris entre 0 et 1 qui représente la part de probabilité de défaillance d'un composant imputable à la défaillance de cause commune.

$$Q(Gr) = \beta \times Q_{el} \qquad Q(E_{iS}) = (1\text{-}\beta) \times Q_{el} \qquad [11\text{-}2]$$

Dans le cas de l'allocation de redondance, les n composants formant un équipement sont identiques. Ils sont donc sujets à une défaillance de cause commune. La redondance active initiale est alors modifiée en prenant en compte une DCC de type β-facteur. La Figure 11.2 permet d'avoir une représentation d'une DCC pour un groupe (*Gr*) de *n* événements de base en redondance active.

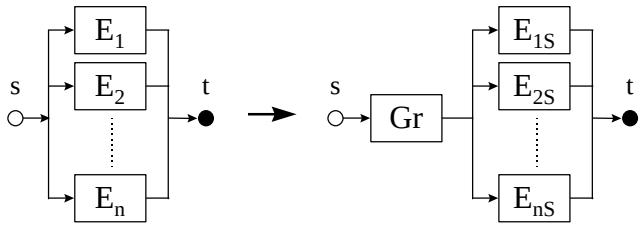


Figure 11.2 : Prise en compte d'une défaillance de cause commune pour des composants en redondance active
La défiabilité de l'équipement fait intervenir le facteur β_n de la DCC.

$$Q_S = ((1\text{-}\beta_n) \times Q_{el})^n + \beta_n \times Q_{el} \qquad [11\text{-}3]$$

On remarque que le facteur du modèle de DCC est dépendant du niveau de redondance. Les retours d'expérience sur les défaillances de cause commune décrits dans [Lib96] (Cf. § 8.2.2) montrent qu'en général, il faut fixer β_2 à 0,03 et β_3 à 0,01. Dans ce cas on retrouve approximativement les résultats du Tableau 8-1.

11.3.3 REPRESENTATION DE LA FONCTION DE COUT ASSOCIEE A UNE ARETE

Pour chaque arête, la fonction de coût (coût en fonction de la probabilité de défaillance) est une fonction en escalier représentée Figure 11.3 pour $C_{el} = 10$; $Q_{el} = 0{,}1$; $min = 1$; $max = 4$ l ; $\beta_2 = 0{,}03$; $\beta_3 = 0{,}01$; $\beta_4 = 0{,}008$

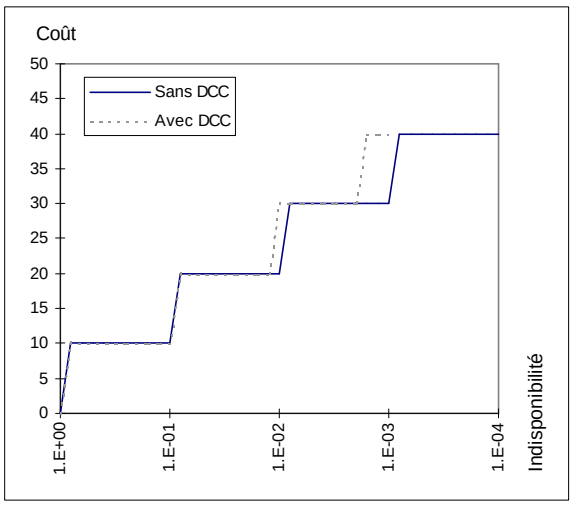


Figure 11.3 : Représentation d'une fonction de coût de type redondance

Comme prévu, et comme on peut le remarquer sur la Figure 11.3, les défaillances de cause commune réduisent de manière drastique les avantages fiabilistes d'une redondance d'ordre important.
La valeur de référence (système à améliorer ou retour d'expérience) de chaque variable de décision est égale à la moyenne minorée entre *min* et *max* (si *min* = 1 et *max* = 4, alors *choix* = 2). La probabilité ($P_{Réf}$) et le coût ($C_{Réf}$) de référence du système sont obtenus en fixant toutes les variables de décision aux valeurs de référence.

11.3.4 MISE EN ŒUVRE DE L'ETUDE

La démarche est donc la suivante :

- Pour chaque arête α du réseau,
  - Création d'une variable de décision entière *choix*(α), bornée sur l'intervalle [*min*;*max*],
  - Création d'une variable intermédiaire réelle *coût*(α), égale à $C_{el} \times choix(\alpha)$
  - Création d'une variable intermédiaire réelle *proba*(α), égale à $(Q_{el})^{choix(\alpha)}$.
  - Affectation de la probabilité de la variable α du réseau de fiabilité à la valeur *proba*(α).
- Création d'une variable intermédiaire *Coût*, égale à la somme des *coût*(α).
- Création d'une variable intermédiaire *Proba*, égale à la probabilité d'occurrence de la cible principale du réseau de fiabilité considéré (au temps t=0 : les probabilités étant constantes).

Pour chacun des problèmes considérés, une première solution a été déterminée à l'aide de l'algorithme de descente rapide. Cette solution a servi de borne de départ pour l'algorithme du Branch & Bound. Avoir une première solution satisfaisante pour l'algorithme du Branch & Bound permet un gain important de temps de calcul, puisque dès le début les branches ne permettant pas d'améliorer la solution courante ne seront pas explorées.
Dans tous les cas, le temps de calcul pour l'algorithme du branch&bound a été limité à une heure ; la durée d'attente acceptable pour un ingénieur fiabiliste. Au delà d'une heure, l'algorithme est interrompu et retourne la meilleure solution en cours.
Les paramètres de l'étude ont été fixés pour tous les réseaux de fiabilité aux valeurs suivantes :

- $C_{el} = 10$, $Q_{el} = 0.01$: caractéristiques élémentaires des composants constituant les équipements ;
- $min = 1$, $max = 3$: bornes minimum et maximum du nombre de composants en redondance ;
- *CoefC* = 0.85, *CoefQ* = 0.5 : coefficients de dégradation du coût ou de la fiabilité de l'installation ;
- $\beta_2 = 0.03$, $\beta_3 = 0.01$: coefficients de défaillance de cause commune pour 2 et 3 composants.

11.3.5 RESULTATS

Les résultats de l'allocation de redondance avec et sans DCC sont présentés respectivement aux Tableau 11-2 et Tableau 11-3.
Quelques explications supplémentaires sont nécessaires pour comprendre ces tableaux.

- L'astérisque * située à droite de certains résultats (coût ou probabilité) précise que la solution fournie par l'algorithme de Descente rapide n'a pas abouti à l'optimum du problème. La solution a donc été trouvée par l'algorithme du Branch&Bound.
- La colonne Q1 correspond au gain relatif en coût ($(C_{Ref}-C_1) / C_{Ref} \times 100$) pour passer du coût de référence à un coût optimisé pour une probabilité inférieure ou égale à la probabilité de référence ($Proba \leq P_{Ref}$).
- La colonne Q2 correspond à la perte relative en coût ($(C_2 - C_1) / C_1 \times 100$) pour passer du coût optimisé pour $Proba \leq P_{Ref}$ à un coût optimisé pour $Proba \leq P_{Ref} \times CoefQ$.
- La colonne Q3 correspond au facteur (P_{Ref} / P_3) entre la probabilité optimisée pour un coût inférieur ou égal au coût de référence et la probabilité de référence : gain d'un facteur x.
- La colonne Q4 correspond au facteur (P_4 / P_3) entre la probabilité optimisée pour un $Coût \leq C_{Ref} \times CoefC$ et la probabilité optimisée pour un $Coût \leq C_{Ref}$: perte d'un facteur x.

	Cost					Q1	Q2	Proba					Q3	Q4
Net	CRef	C1	Tps1	C2	Tps2	Gain	Perte	PRef	P3	Tps3	P4	Tps4	Gain	Perte
1	100	100	0.3	110	0.4	0.0	10.0	2.00E-08	1.02E-08	0.3	1.02E-06	0.2	2	100
2	160	150	0.9	160	0.9	6.3	6.7	2.00E-12	2.04E-14	0.8	1.03E-10	0.7	98	5050
3	160	150	1.1	150	0.8	6.3	0.0	3.00E-08	5.03E-10	0.7	5.01E-08	0.6	60	100
4	180	160	1.0	170	1.1	11.1	6.3	2.00E-08	2.03E-10	1.2	2.03E-08	1.2	99	100
5	240	190	1.6	200	1.9	20.8	5.3	3.00E-08	1.05E-10 *	1.3	1.05E-08	1.7	286	100
6	300	220	3.4	220	2.9	26.7	0.0	1.00E-12	1.02E-18	2.2	1.07E-16	6.6	980100	105
7	420	280	2.8	290	3.9	33.3	3.6	6.00E-12	6.01E-18 *	6.5	5.31E-16 *	4.0	998002	88
8	260	180	0.9	190	1.0	30.8	5.6	3.00E-08	3.06E-12 *	2.8	2.07E-10 *	1.8	9804	68
9	240	190	1.4	200	1.3	20.8	5.3	2.00E-08	6.04E-12 *	1.2	6.04E-10	1.0	3312	100
10	240	190	1.7	190	1.5	20.8	0.0	1.00E-08	1.06E-12 *	1.7	3.03E-10	1.9	9437	286
11	240	200	3.9	210	3.2	16.7	5.0	2.00E-12	3.07E-16 *	1.4	1.04E-12	3.4	6515	3389
12	260	230	10.3	240	5.6	11.5	4.3	3.00E-12	2.05E-14 *	3.2	3.08E-12	10.3	146	150
13	580	370	23.8	370	10.1	36.2	0.0	2.00E-12	2.00E-18 *	406.9	2.19E-18 *	2797.4	1000330	1
14	520	390	557.9	420	1552.4	25.0	7.7	6.01E-08	6.62E-11 *	1374.7	2.76E-09 *	2104.1	907	42
15	280	230	6.8	240	3.6	17.9	4.3	2.00E-08	6.05E-12 *	2.0	1.05E-08 *	5.6	3307	1736
16	420	300	14.9	310	14.6	28.6	3.3	2.00E-12	2.01E-18 *	5.2	2.08E-16 *	5.7	994818	104
17	500	300	5.2	300	4.0	40.0	0.0	1.00E-08	1.00E-12 *	114.1	1.04E-12 *	634.9	10004	1
18	440	280	3.4	280	3.0	36.4	0.0	1.00E-08	1.00E-12 *	20.1	1.03E-12 *	33.4	10001	1
19	560	400	338.1	420	519.0	28.6	5.0	6.00E-12	1.11E-17 *	521.1	7.22E-15 *	98.4	540260	650
Max			557.9		1552.4	40.0	10.0			1374.7		2797.4	1000330	5050
Moy.			51.5		112.2	22.0	3.8			129.9		300.7	240394	641
Med.			3.4		3.0	20.8	4.3			2.2		4.0	6515	100

Tableau 11-2 : Allocation de redondance sans défaillance de cause commune

Nous pouvons maintenant répondre effectivement aux quatre questions que se posait le décideur. A partir de la configuration de référence, nous trouvons, à fiabilité identique, des solutions améliorant, en moyenne, le coût de 22% et de 23,2%, selon que l'on prenne ou non en compte les défaillances de cause commune.

A l'exception du réseau de fiabilité SR93-14 avec prise en compte des défaillances de cause commune, l'algorithme du Branch&Bound met moins d'une heure pour résoudre tous ces problèmes. Il est logique, vue sa structure que le réseau SR93-14 pose un problème lors d'une recherche exhaustive. Parmi les nombreux équipements en série (branche inférieure menant à la destination), aucun n'est prédominant, ce qui oblige l'algorithme à considérer toutes les combinaisons de redondance les concernant. Ces équipements étant de même poids, tant pour le coût que pour la fiabilité du système, il suffirait d'étudier une seule combinaison. Pour réduire de manière efficace la durée de l'algorithme, il suffit de prétraiter le sous-système série en le considérant dans son ensemble. Le nombre de solutions intermédiaires passe alors de $3^5 = 243$ à $(3-1)\times5+1=11$.

Les gains obtenus en termes de fiabilité sont en revanche assez irréalistes lorsque les défaillances de cause commune sont ignorées. Gagner un facteur 10^{+6} sur 5 des 19 réseaux de fiabilité sans dégrader le coût global de l'installation en prenant 0,01 comme défiabilité des composants de base montre effectivement qu'il doit y

avoir un problème dans la modélisation. La prise en compte des DCC permet de ramener ce facteur à une valeur raisonnable (environ 60).

	Cost					Q1	Q2	Proba					Q3	Q4
Net	CRef	C1	Tps1	C2	Tps2	Gain	Perte	PRef	P3	Tps3	P4	Tps4	Gain	Perte
1	100	100	0.3	110	0.3	0.0	10.0	3.11E-07	1.97E-07	0.3	4.14E-06	0.4	1.58	21.00
2	160	150 *	0.7	160	0.6	6.3	6.7	1.23E-10	3.20E-11 *	0.6	1.65E-09	0.9	3.83	51.47
3	160	140	0.8	150 *	0.7	12.5	7.1	4.66E-07	1.23E-07 *	0.7	5.84E-07	0.8	3.79	4.75
4	180	150	1.0	160	0.9	16.7	6.7	3.11E-07	5.24E-08	0.7	2.47E-07	1.0	5.94	4.71
5	240	190	2.0	200	1.8	20.8	5.3	4.66E-07	3.57E-08	1.2	2.14E-07	1.9	13.07	6.01
6	300	210	3.1	220	3.1	30.0	4.8	6.13E-11	1.03E-12 *	2.1	1.34E-12 *	2.4	59.34	1.30
7	420	290	6.4	290 *	6.0	31.0	0.0	3.67E-10	6.18E-12 *	4.0	9.79E-12 *	5.2	59.42	1.58
8	260	190	2.3	200	2.2	26.9	5.3	4.66E-07	3.08E-08 *	1.4	8.07E-08 *	1.7	15.13	2.62
9	240	190	1.9	200	1.8	20.8	5.3	3.11E-07	2.21E-08 *	1.3	1.21E-07 *	1.8	14.07	5.46
10	240	180	2.1	190	2.1	25.0	5.6	1.56E-07	1.08E-08 *	1.7	1.72E-08 *	1.7	14.40	1.59
11	240	200	1.5	210	1.6	16.7	5.0	1.23E-10	8.36E-12	1.3	8.61E-11	1.6	14.66	10.30
12	260	230 *	2.3	250	2.3	11.5	8.7	1.84E-10	3.62E-11	4.3	1.89E-10	4.8	5.08	5.21
13	580	350	17.0	360 *	20.7	39.7	2.9	1.23E-10	2.06E-12 *	10.8	2.07E-12 *	24.9	59.52	1.01
14	520	430	3697.9	420 *	3703.7	17.3	-2.3	9.35E-07	8.56E-08 *	3681.4	2.99E-07 *	3683.4	10.92	3.49
15	280	220	3.6	230 *	3.5	21.4	4.5	3.11E-07	2.22E-08 *	2.6	1.34E-07	3.5	14.03	6.06
16	420	290	7.0	300	7.4	31.0	3.4	1.23E-10	2.06E-12 *	5.1	2.48E-12 *	5.6	59.43	1.20
17	500	290 *	12.7	300 *	12.6	42.0	3.4	1.56E-07	1.02E-08 *	92.9	1.02E-08 *	982.8	15.25	1.00
18	440	260	8.2	280	9.2	40.9	7.7	1.55E-07	1.02E-08 *	19.2	1.02E-08 *	15.1	15.24	1.00
19	560	390 *	42.5	410 *	34.9	30.4	5.1	3.67E-10	6.33E-12 *	37.2	2.97E-11 *	52.9	58.04	4.70
Max			3697.9		3703.7	42.0	10.0			3681.4		3683.4	59.52	51.47
Moy.			200.7		200.8	23.2	5.0			203.6		252.2	23.30	7.08
Med.			2.3		2.3	21.4	5.3			2.1		2.4	14.40	4.70

Tableau 11-3 : Allocation de redondance avec défaillance de cause commune

Ces deux tableaux permettent d'avoir une synthèse des résultats, mais ils ne permettent pas d'appréhender la qualité des résultats. Pour ce faire, il convient d'étudier les solutions des allocations. En travaillant sur une description visuelle des systèmes étudiés et sur des variables de décision entières, il est possible de représenter graphiquement les résultats d'une affectation. Dans la Figure 11.4, les niveaux de redondance des arêtes sont représentés par l'épaisseur des arêtes (Cf. légende de la figure).

Deux stratégies permettent l'amélioration des réseaux de fiabilité :

- En jouant sur les arêtes appartenant aux coupes les plus courtes, c'est-à-dire les points faibles du réseau
- En jouant sur les arêtes appartenant aux chemins les plus courts, c'est-à-dire les points forts du réseau

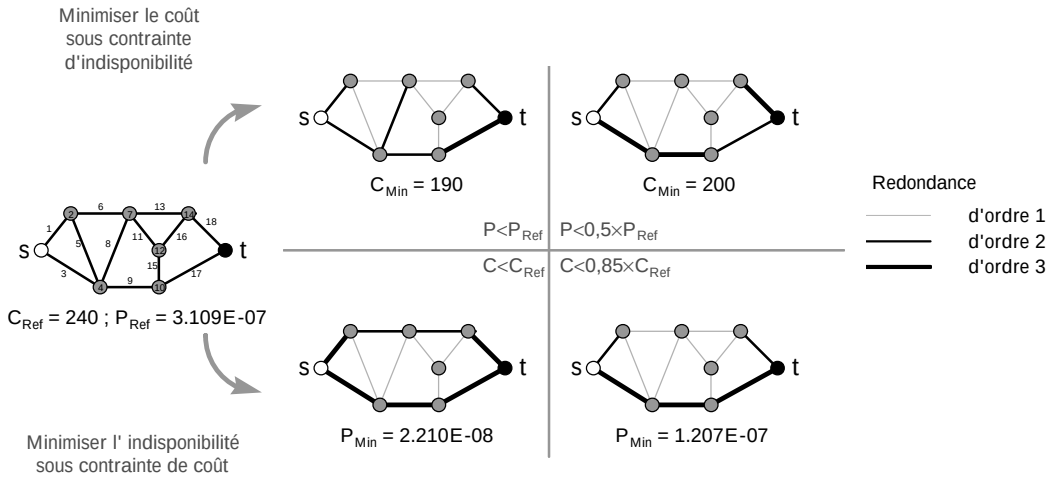


Figure 11.4 : Représentation des résultats pour SR93-9 en prenant en compte les DCC

Si on s'intéresse à la minimisation de l'indisponibilité sous contrainte de coût (voir Figure 11.4), on remarque que pour les deux contraintes de coût, une redondance d'ordre 3 est nécessaire sur le chemin le plus court. Pour la plus forte contrainte de coût, une redondance d'ordre 2 sur les deux autres arêtes appartenant aux

coupes d'ordre 2 est mise en œuvre. Pour la contrainte de coût la plus faible, ces deux mêmes arêtes bénéficient d'une redondance d'ordre 3 et les autres arêtes appartenant au deuxième plus court chemin sont ensuite favorisées.
Concernant la minimisation du coût sous contrainte d'indisponibilité, les résultats sont beaucoup plus chaotiques. Considérons par exemple la Figure 11.4 : lorsque la contrainte d'indisponibilité est la plus forte, l'allocation fournit une solution optimale d'un point de vue du coût (il n'y a pas de solution à moindre coût répondant à la contrainte fiabiliste), mais pas d'un point de vue de l'indisponibilité. Il existe des solutions à un coût égal à 200 avec de meilleures qualités fiabilistes. Dans notre problème, le coût est discrétisé, c'est-à-dire que pour un coût donné, il existe une grande quantité de configurations possibles. Or les algorithmes mis en œuvre (Descente rapide et Branch & Bound) renvoient la première affectation qui optimise la grandeur considérée tout en satisfaisant les contraintes du problème.

Le coût étant discrétisé, il est possible pour chaque seuil de coût possible de calculer l'indisponibilité optimisée du système. Cette information a été calculée avec et sans prise en compte des DCC. Elle est représentée sur la Figure 11.5 sous la forme d'une courbe.

Cout	Sans DCC	Avec DCC
120	2.050E-04	2.050E-04
130	1.050E-04	1.080E-04
140	5.049E-06	1.099E-05
150	2.050E-06	6.235E-06
160	1.050E-06	2.440E-06
170	5.049E-08	4.710E-07
180	2.050E-08	3.256E-07
190	1.050E-08	1.808E-07
200	6.040E-10	1.207E-07
210	4.040E-10	8.509E-08
220	2.050E-10	5.433E-08
230	1.051E-10	2.472E-08
240	6.040E-12	2.210E-08
250	4.040E-12	2.150E-08
260	2.051E-12	2.090E-08
270	2.040E-12	2.073E-08
280	2.030E-12	2.056E-08
290	2.020E-12	2.046E-08
300	2.010E-12	2.043E-08
310	2.001E-12	2.041E-08
320	2.000E-12	2.041E-08
330	2.000E-12	2.040E-08
340	2.000E-12	2.040E-08
350	2.000E-12	2.040E-08
360	2.000E-12	2.040E-08

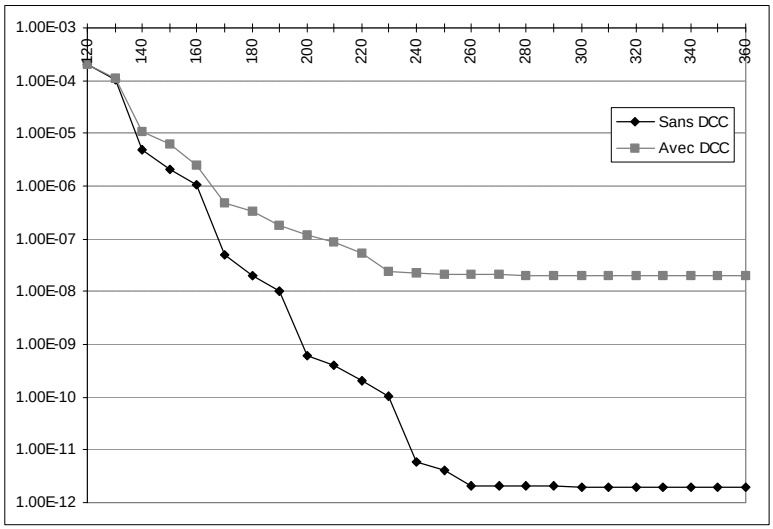


Figure 11.5 : Indisponibilité optimisé en fonction du coût pour SR93-9

Cette courbe est une réelle source d'informations pour le décideur. Elle permet de visualiser combien coûte l'installation pour une disponibilité donnée et, inversement, comment évolue la fiabilité du système pour un coût donné.

On remarque qu'au delà d'une certaine limite, les améliorations que l'on peut apporter au système, ne permettent pas d'améliorer de manière significative sa fiabilité. Les défaillances de cause commune limitent les possibilités d'amélioration de la fiabilité du réseau.
Les stratégies mises en œuvre par les deux modélisations du problème (avec ou sans défaillances de cause commune) ne sont pas tout à fait les mêmes, comme on peut le constater sur la Figure 11.6. La partie supérieure de la figure correspond aux résultats du problème ne prenant pas en compte les DCC.

Lorsque les défaillances de cause commune sont négligées, l'algorithme cherche avant tout à fiabiliser le chemin le plus court (pour un coût de 150 à 180) en commençant par les arêtes les plus critiques (160, 170, 190, 200) c'est-à-dire celles qui appartiennent aux coupes minimales les plus courtes.
La stratégie prenant en compte les défaillances de cause commune cherche à améliorer, dans une première phase, les arêtes critiques (150, 160, 180, 190) en commençant par celles qui appartiennent au chemin le plus court, puis, dans une seconde phase, les chemins les plus courts (170, 200).

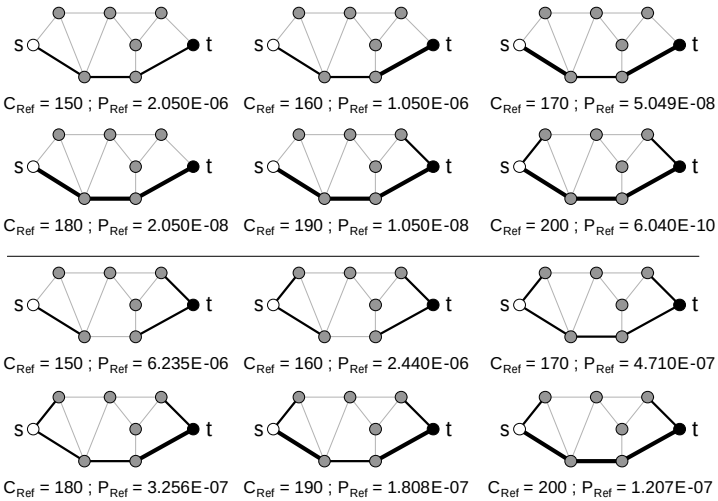


Figure 11.6 : Comparaison d'allocations avec et sans DCC pour SR93-9

11.3.6 CONCLUSION

Nous avons vu dans ce chapitre que les défaillances de cause commune pouvaient être assez facilement prises en compte dans les problèmes d'allocation de redondance et qu'elles jouaient un rôle non négligeable quant aux résultats de l'allocation.
Les résultats obtenus sont bien conformes à une optimisation que l'on pourrait faire intuitivement à la main. Il est clair que chercher à améliorer en priorité les composants critiques et les chemins de succès d'un système est une stratégie qui n'a pas besoin d'outils pour être mise en œuvre.

11.4 OPTIMISATION DE LA MAINTENANCE

Il existe de nombreuses politiques de maintenance. Parmi celles-ci :

- La maintenance corrective qui consiste à ne réparer un composant que lorsqu'il est en panne et qu'il a été détecté comme tel. Elle suppose des tests périodiques pour les composants fonctionnant sur sollicitation, un nombre important de réparateurs (surtout sur des systèmes de production pour lesquels la disponibilité est cruciale) et une bonne gestion des pièces de rechange.
- La maintenance préventive qui consiste à changer périodiquement les composants, soit uniquement ceux en panne (comme les néons des lieux publiques), soit de manière systématique. Les périodes sont généralement déterminées en fonction du MTTF (Mean Time To Failure) des composants considérés.
- La maintenance prédictive qui consiste à changer un composant juste avant qu'il ne tombe en panne. Cette politique s'appuie sur le fait qu'un composant donne généralement des signes d'usure avant de tomber en panne. Il convient donc de bien interpréter ces signes afin de changer les pièces nécessaires avant leur panne.

11.4.1 PRINCIPE RETENU

Pour notre optimisation de la maintenance, nous considérons que toutes les arêtes des réseaux sont des composants remplacés périodiquement. Le remplacement est supposé instantané et fiable (pas de défaillance possible à la remise en route). La disponibilité de chaque composant est donc modélisé par la loi de probabilité suivante :

$$Q(t) = 1 - \exp^{-l(t \bmod T)} \qquad [11\text{-}4]$$

où l est le taux de défaillance du composant et T la période de remplacement.

Le but étant d'optimiser la maintenance, nous considérons que le taux de défaillance est le même pour tous les composants : l.

D'un point de vue maintenance, nous pouvons jouer sur les périodes de remplacement des composants. Plus le composant est crucial pour le fonctionnement du système, plus il est préférable de le changer fréquemment afin que le système soit le moins possible à l'arrêt.
Pour chaque composant, nous considérons un choix de N périodes de remplacement (T_i avec i de 1 à n). Le coût du remplacement d'un composant est égal à C_R.
A partir d'une période d'observation suffisamment longue *TM*, nous pouvons calculer pour chaque composant le nombre de remplacements effectués et donc le coût associé à la maintenance du système.
Pour contrebalancer ce coût, il convient aussi d'avoir une information sur le coût d'indisponibilité du système. Lorsqu'un système de production fonctionne, il engendre des gains financiers. Dans le cas contraire, en plus de la perte de production, des réclamations clients et des "retours ateliers" qui correspondent à des coût directs (chiffrables), viennent s'ajouter des coûts indirects tels que la publicité négative ou des pertes de contrat, qui eux ne sont pas chiffrables aisément.
Le coût d'indisponibilité du système peut être défini comme étant le produit d'un coefficient C_{IH} et du temps d'indisponibilité sur la période d'observation considérée. Le temps d'indisponibilité correspond à l'indisponibilité moyenne du système multipliée par la période d'observation. Nous approximons l'indisponibilité moyenne à partir d'une intégrale numérique de 0 à *TM* de son indisponibilité instantanée. Pour cela nous fixons le pas du calcul de l'intégrale à *step* : paramètre de l'étude.
L'objectif est maintenant de minimiser le coût d'indisponibilité du système et son coût de maintenance, ou plus simplement la somme de ces deux coûts.

Les paramètres de l'étude sont donc :

T_i (pour i de 1 à N), C_R, l : identique pour chaque arête du réseau

TM, C_{IH} , *step* pour l'étude du système.

Un dernier paramètre *Ref* (compris entre 1 et *N*) correspond à la valeur de la variable de décision de chaque arête du réseau permettant de calculer la probabilité ($P_{Réf}$) et le coût ($C_{Réf}$) de référence du système.

11.4.2 MISE EN ŒUVRE DE L'ETUDE

Aralia dispose d'une loi de probabilité du type "test périodique" comportant 3 paramètres : le taux de défaillance, la période de test et la date du premier test. Ce dernier est fixé à zéro. Le taux de défaillance est égal à l pour tous les composants.
Bien qu'il soit possible, à l'aide des opérateurs du langage Aloès et de la connexion avec Aralia, de calculer l'indisponibilité moyenne, nous avons préféré, pour des raisons de performances, inclure directement le calcul des valeurs moyennes de grandeurs fiabilistes dans l'interface entre Aralia et Aloès. Les gains en termes de temps de calcul sont de l'ordre d'un facteur 7 à 8. Cette différence est principalement due au nombre de communications utiles entre Aloès et Aralia.

La démarche est donc la suivante :

- Pour chaque arête α du réseau,
  - Création d'une variable de décision entière *choix*(α), bornée sur l'intervalle [*1*;*N*]
  - Création d'une variable intermédiaire réelle *période*(α), égale à $T_{choix(\alpha)}$
  - Création d'une variable intermédiaire réelle *coût*(α), égale à **Int**(*TM* / *période*(α)) × C_r (où **Int** est une fonction renvoyant la valeur entière de son paramètre), c'est-à-dire le nombre de remplacements du composant α sur le temps de mission considéré, multiplié par le coût de son remplacement.
  - Affectation d'une loi test périodique de paramètre (l, *période*(α), 0) à la variable α du réseau de fiabilité.
- Création d'une variable intermédiaire *IndispoMoy*, égale à l'indisponibilité moyenne de la cible principale du réseau de fiabilité considéré (de t = 0 à t = *TM* par pas de *step*).
- Création d'une variable intermédiaire *CoûtMaintenance* égale à la somme des *coût*(α).
- Création d'une variable intermédiaire *Coût* = *CoûtMaintenance* + *IndispoMoy* × *TM* × C_{IH}

Dans cette modélisation, l'algorithme du Branch & Bound ne pourra faire aucune séparation s'appuyant sur le coût du système, car ce dernier est *a priori* non-monotone en fonction des variables de décision. En effet, il est défini comme la somme de deux grandeurs, le coût de maintenance et le coût de l'indisponibilité du

système, qui sont toutes les deux monotones en fonction des variables de décision. Mais l'une est croissante et l'autre décroissante.

Le temps de calcul pour l'algorithme du Branch & Bound a aussi été limité à 10 minutes (600 secondes).
Les paramètres de l'étude ont été fixés pour tous les réseaux de fiabilité à

- $N = 4$, $T_1 = 10$, $T_2 = 20$, $T_3 = 40$, $T_4 = 80$: les différentes périodes de remplacement ;
- $C_r = 10$, $l = 0.01$: les caractéristiques élémentaires des composants ;
- $TM = 500$, $C_{IH} = 50$, $step = 2.5$: le temps de mission, le coût d'indisponibilité horaire et le pas pour le calcul de l'indisponibilité moyenne ;
- $CoefC = 0.85$, $CoefQ = 0.5$: les coefficients de dégradation du coût ou de la fiabilité de l'installation.

11.4.3 RESULTATS

	Cost					Q1	Q2	Proba					Q3	Q4
Net	CRef	C1	Tps1	C2	Tps2	Gain	Perte	PRef	P3	Tps3	P4	Tps4	Gain	Perte
1	1991	1954	9.2	2168	4.6	1.8	10.9	2.96E-02	2.38E-02	6.5	3.11E-02	14.9	19.7	30.9
2	2123	1982 *	163.4	2487	76.4	6.7	25.5	4.94E-03	4.07E-03 *	125.1	6.45E-03 *	228.9	17.6	58.5
3	3079	2843	140.8	3194 *	27.1	7.7	12.4	4.32E-02	2.35E-02	22.4	3.65E-02	618.6	45.6	55.6
4	3057	2761	567.3	3191 *	115.7	9.7	15.6	3.23E-02	1.75E-02 *	71.8	4.18E-02	625.7	45.6	138.0
5	4236	3642	646.2	4095 *	624.4	14.0	12.4	4.95E-02	2.30E-02 *	363.8	5.07E-02	652.1	53.6	120.9
6	3829	2727	690.7	3388	666.5	28.8	24.3	3.16E-03	1.22E-03	656.8	2.82E-03	672.5	61.3	131.1
7	5537	3487	854.4	4463	784.0	37.0	28.0	1.15E-02	4.73E-03	733.5	7.31E-03	773.2	58.8	54.5
8	4302	3002	673.7	3742	639.8	30.2	24.6	4.21E-02	1.60E-02 *	485.5	2.33E-02	650.9	62.0	45.8
9	3904	3396	642.4	4046	625.1	13.0	19.2	3.62E-02	1.86E-02 *	631.3	5.17E-02	666.0	48.6	178.0
10	3620	3090	641.0	3654	625.6	14.6	18.3	2.48E-02	1.24E-02 *	631.4	2.32E-02	650.4	50.0	87.1
11	3135	2733	639.3	3235	632.3	12.8	18.4	5.40E-03	3.06E-03	634.4	6.62E-03	644.8	43.3	116.3
12	3452	3122	644.1	3827	632.6	9.6	22.6	8.09E-03	7.06E-03	642.0	1.11E-02	656.5	12.7	57.4
13	7424	4854	1124.2	6217	1001.1	34.6	28.1	6.97E-03	2.64E-03	932.8	7.38E-03	1024.0	62.1	179.5
14	9772	8535	900.4	10055	723.3	12.7	17.8	1.31E-01	7.07E-02	769.6	1.33E-01	911.3	46.0	88.8
15	4450	3855	661.0	4705	631.0	13.4	22.0	3.80E-02	2.25E-02	644.2	5.22E-02	675.2	40.7	131.6
16	5388	4027	842.7	4649	796.7	25.3	15.4	5.52E-03	2.46E-03	767.0	5.85E-03	811.0	55.5	138.1
17	6835	4889	963.8	6320	822.6	28.5	29.3	2.34E-02	9.32E-03	839.8	1.67E-02	925.9	60.2	79.1
18	5990	4248	867.2	5394	772.3	29.1	27.0	1.96E-02	8.86E-03	763.5	1.37E-02	821.2	54.8	54.3
19	7417	6175	1199.6	7425	1038.6	16.7	20.2	1.67E-02	9.52E-03	1106.6	1.79E-02	1291.0	42.9	88.2
Max			1199.6		1038.6	37.0	29.3			1106.6		1291.0	62	180
Moy.			677.4		591.6	18.2	20.6			569.9		700.7	46	97
Med.			661.0		632.6	14.0	20.2			642.0		666.0	49	88

Tableau 11-4 : Résultats de l'optimisation de maintenance

Une synthèse des résultats est donnée au Tableau 11-4 dans le même format que celle du chapitre précédent . Le gain Q3 et la perte Q4 sont exprimés ici sous la forme d'une valeur relative et non d'un facteur.

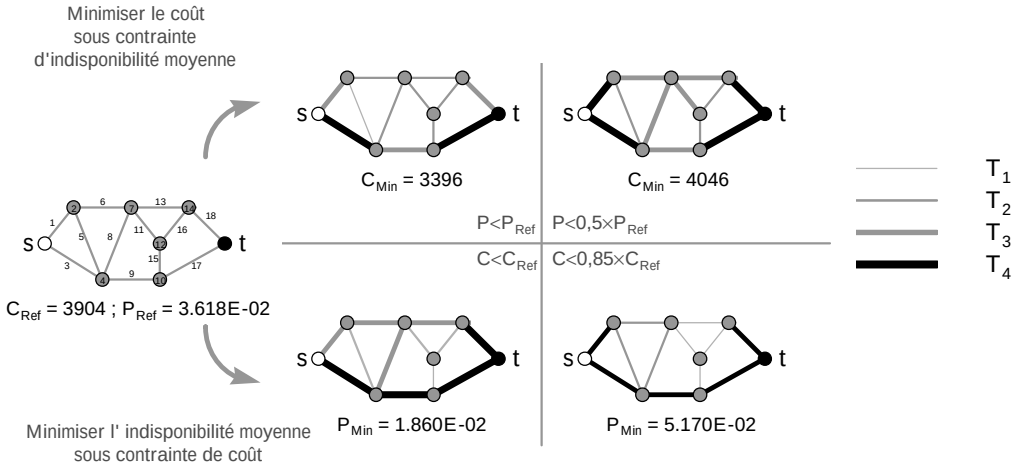


Figure 11.7 : Représentation graphique des allocations pour l'optimisation de maintenance de SR93-9

On constate que l'algorithme du Branch & Bound n'aboutit que pour les réseaux de fiabilité de petite taille (jusqu'à SR93-5). En revanche, les rares fois où il aboutit et dans la moitié des cas, il trouve une solution meilleure que l'algorithme de Descente rapide. L'algorithme du Branch & Bound n'est pas assez rapide, mais l'algorithme de Descente rapide n'est donc pas assez robuste pour ce type de modélisation.

Les stratégies d'allocation sont beaucoup plus difficiles à cerner sur ce type de modèles et ce, pour de nombreuses raisons :

- La variable de décision peut prendre quatre valeurs (au lieu de trois dans le cas de l'allocation de redondance traitée au chapitre précédent)
- La variation entre deux échelons n'a pas la même répercussion sur l'indisponibilité du système
- Il existe une corrélation entre la valeur à optimiser et la contrainte du problème.

Les règles générales décrites au chapitre précédent restent tout de même valables ce qui permet de valider intuitivement ces résultats.

11.4.4 CONCLUSION

Il est possible, grâce au langage Aloès, de décrire des problèmes complexes comme certaines politiques de maintenance. Par contre les algorithmes présents au sein du logiciel ne semblent pas suffisamment robustes ou rapides pour que l'on puisse se satisfaire des résultats obtenus.
Il conviendra dans l'avenir d'améliorer ces algorithmes, voire d'en ajouter d'autres plus robustes permettant de mieux prendre en compte des grandeurs non monotones en fonction des variables de décision du problème.

11.5 ALLOCATION EN CONCEPTION PRELIMINAIRE

Comme expliqué au chapitre 10.2.1, l'allocation en conception préliminaire se fait de manière plus réaliste si les variables de décision, associées aux composants, sont des variables de décision continues. En effet le niveau de détail de l'analyse, lors de cette phase, s'arrête aux sous-systèmes. Les grandeurs fiabilistes associées aux sous-systèmes ne peuvent pas évoluer de manière discrète. Le résultat d'une allocation sera utilisé par la suite comme objectif d'une allocation plus détaillée sur chacun des sous-systèmes. On peut alors associer à ces derniers une variable de décision représentant leur probabilité ou leur taux de défaillance. Une fonction d'effort précise alors la difficulté inhérente à l'amélioration de la fiabilité de ce composant.

Nous avons vu au chapitre 9.2.3, que parmi les fonctions de coût continues de la littérature, seule la fonction de Truelove était réellement intéressante. En effet, les autres sont, soit impossibles à régler, soit ne répondant pas à une réalité physique, soit un cas particulier ou une complication inutile de la fonction de Truelove.
Pour rappel, cette dernière est définie par l'équation suivante :

$$C_{Tr}(R) = a \times \frac{1}{(1-R)^b} \qquad [11\text{-}5]$$

où a et b sont des réels strictement positifs.
Cette fonction est suffisamment paramétrable pour passer par deux points significatifs.

Dans cette expérimentation, la fonction d'effort de chaque sous-système sera exprimée à l'aide de la fonction de Truelove et sera définie de telle manière qu'elle passe par un point significatif provenant du retour d'expérience : $\{Q_{REX} ; C_{REX}\}$. Dans ce cas b sera fixé à 1.
Le coût du système est défini comme étant la somme des coûts des composants.
La probabilité ($P_{Réf}$) et le coût ($C_{Réf}$) de référence du système sont définis en prenant comme valeurs pour chaque composant son coût et sa probabilité de retour d'expérience.
Les paramètres de l'étude sont donc Q_{REX} et C_{REX}.

11.5.1 MISE EN ŒUVRE DE L'ETUDE

La démarche de génération du modèle à partir du réseau de fiabilité et des paramètres de l'étude est donc la suivante :

- Pour chaque arête α du réseau,
  - Création d'une variable de décision réelle *choix*(α), bornée sur l'intervalle [0;1],
  - Création d'une variable intermédiaire réelle *coût*(α), égale à $C_{REX} \times Q_{REX}$ / *choix*(α).
  - Affectation de la probabilité de la variable α du réseau de fiabilité à la valeur *choix*(α).
- Création d'une variable intermédiaire *Coût*, égale à la somme des *coût*(α).
- Création d'une variable intermédiaire *Proba*, égale à la probabilité d'occurrence de la cible principale du réseau de fiabilité considéré (au temps t=0 : les probabilités étant constantes).

Le seul algorithme actuellement disponible pour traiter les problèmes faisant intervenir des variables de décision continues est l'algorithme du Simplex. De nombreux paramètres sont disponibles. Il est donc relativement complexe de mettre en place un plan d'expérience qui permette de définir les valeurs optimales de ces paramètres.

Les paramètres utilisés pour la réflexion, la contraction et l'extension (α ; β , γ) ont été fixé (-1 ; 0,5 ; 2), valeurs conseillées par Nelder et Mead [NM65]. Le nombre maximum de cycles de l'algorithme a été fixé à 10, ce qui permet, d'après des études préliminaires, d'atteindre une valeur proche de l'optimum. Le paramètre fixant le critère d'arrêt d'un cycle, ε, a été fixé à 10^{-3}. Deux fonctions de pénalisation permettant de prendre la contrainte du problème en compte ont été testées successivement. La première, définie à l'aide de l'équation [10-6] avec un poids fixé à 10^{+7} ($W_i = 10^{+7}$, $\forall i$), a l'avantage de fonctionner même si la contrainte n'est pas respectée. Elle permet de fournir une solution satisfaisant, *a priori*, la contrainte. Le résultat de cette première allocation est utilisé comme affectation initiale d'une seconde allocation prenant une fonction de pénalisation discontinue définie à l'aide de l'équation [10-4] avec un poids de 10^{+10} ($W = 10^{+10}$). Cette seconde fonction de pénalisation ne fonctionne que si l'affectation en cours respecte les contraintes.

La première fonction de pénalisation accepte les affectations ne satisfaisant pas la contrainte. Elle génère facilement le premier Simplex, mais ce dernier est très dispersé dans l'espace des solutions. On doit réaliser de nombreuses itérations afin de concentrer le Simplex autour d'un optimum. A l'opposé, la seconde fonction de pénalisation force le Simplex initial à satisfaire les contraintes du problème ce qui prend en général plus de temps puisque les solutions ne les satisfaisant pas sont rejetées. Le simplex ainsi généré est plus "tassé". Le nombre d'itérations est moins important.

Les paramètres de l'étude ont été fixés pour tous les réseaux de fiabilité à

- $Q_{REX} = 0{,}01$, $C_{REX} = 30$: la valeur du retour d'expérience sur ces sous-systèmes ;
- *CoefC* = 0,85, *CoefQ* = 0,5 : les coefficients de dégradation du coût ou de la fiabilité de l'installation.

11.5.2 RESULTATS

			C1				C2				P3				P4			
Net	CRef	PRef	C(P1)	C(P2)	T(P1)	T(P2)	C(P1)	C(P2)	T(P1)	T(P2)	P(P1)	P(P2)	T(P1)	T(P2)	P(P1)	P(P2)	T(P1)	T(P2)
1	150.00	2.020E-04	130.21	127.92	1.04	0.66	181.10	179.04	1.15	0.82	1.452E-04	1.451E-04	1.27	1.26	2.111E-04	2.059E-04	1.70	1.32
2	240.00	2.040E-06	200.94	195.93	2.42	1.49	251.75	244.57	2.58	1.59	1.219E-06	1.099E-06	2.91	2.91	6.382E-06	1.855E-06	3.24	3.74
3	240.00	3.020E-04	178.43	178.29	2.42	1.59	248.01	247.57	2.80	1.65	1.622E-04	1.605E-04	3.13	2.74	2.874E-04	2.375E-04	3.30	2.80
4	270.00	2.030E-04	181.94	181.51	3.07	1.98	248.70	247.94	3.24	1.98	8.261E-05	8.224E-05	3.57	3.46	1.667E-04	1.257E-04	4.67	3.63
5	360.00	3.059E-04	232.59	229.26	5.54	3.46	319.20	316.42	6.27	3.35	1.168E-04	1.126E-04	7.96	6.05	9.083E-04	4.696E-04	6.48	6.81
6	450.00	1.031E-06	211.55	210.52	8.13	5.06	259.78	257.50	8.62	5.06	6.157E-08	6.093E-08	10.65	9.61	2.178E-07	1.180E-07	9.89	10.44
7	630.00	5.990E-06	276.01	272.92	18.73	10.27	353.57	344.57	17.58	10.60	4.258E-07	3.735E-07	24.94	20.87	6.759E-06	2.103E-06	30.26	22.36
8	390.00	3.010E-04	193.64	192.69	6.59	3.79	269.39	266.90	8.29	3.96	6.636E-05	6.535E-05	8.35	7.80	2.636E-04	1.910E-04	9.44	8.63
9	360.00	2.050E-04	224.09	218.83	5.93	3.41	296.50	295.66	5.71	3.40	6.508E-05	6.435E-05	6.70	6.10	6.170E-04	3.035E-04	7.30	7.64
10	360.00	1.060E-04	199.61	199.09	5.50	3.40	263.42	262.06	5.55	3.41	2.402E-05	2.382E-05	6.86	6.10	5.030E-05	3.861E-05	6.92	6.92
11	360.00	2.050E-06	252.00	249.74	5.16	3.30	316.96	310.43	6.26	3.46	6.260E-07	6.033E-07	7.14	6.59	2.131E-05	1.462E-05	8.46	6.92
12	390.00	3.080E-06	314.32	302.97	6.93	3.90	391.43	377.18	7.25	3.95	1.612E-06	1.563E-06	7.58	7.09	1.471E-05	1.245E-05	9.39	7.14
13	870.00	2.083E-06	333.03	313.61	39.60	23.68	394.97	379.24	34.71	22.41	4.497E-08	4.404E-08	46.91	42.35	2.754E-05	1.726E-05	55.20	46.79
14	780.00	6.541E-04	530.25	507.05	27.52	17.58	700.92	682.76	30.21	16.47	2.795E-04	2.327E-04	32.90	32.19	1.421E-02	9.067E-03	26.58	29.82
15	420.00	2.060E-04	249.51	248.58	8.02	4.78	343.07	336.57	8.13	4.99	6.115E-05	5.954E-05	9.89	8.62	3.074E-03	5.697E-04	11.09	10.99
16	630.00	2.041E-06	289.10	286.25	17.52	10.61	365.08	361.52	18.72	10.93	1.622E-07	1.464E-07	26.31	20.98	3.460E-07	3.157E-07	31.31	21.58
17	750.00	1.041E-04	254.27	243.99	27.58	16.26	312.91	311.29	28.18	16.25	4.953E-06	4.799E-06	39.28	30.37	4.473E-05	3.773E-05	45.04	31.47
18	660.00	1.021E-04	207.18	205.60	18.67	11.59	275.74	269.65	21.42	11.75	4.647E-06	4.545E-06	24.11	22.19	1.991E-04	1.174E-04	25.21	23.89
19	840.00	6.114E-06	443.64	440.88	37.35	22.68	558.16	545.95	45.81	22.90	7.178E-07	6.770E-07	54.55	42.84	4.058E-06	1.976E-06	52.45	47.29

Tableau 11-5 : Résultats de l'allocation en conception préliminaire

Les résultats sont regroupés au Tableau 11-5. Pour chacune des quatre questions (C1, C2, C3, C4) que l'on peut se poser sur un système, quatre résultats (C(P1), C(P2), T(P1), T(P2)) sont affichés :

- les deux premiers sont les valeurs de la grandeur à optimiser en considérant les deux fonctions de pénalisation,
- les deux derniers correspondent aux temps de calcul de l'algorithme pour chaque fonction de pénalisation.

La solution proposée à l'aide de la fonction de pénalisation discontinue est toujours de meilleure qualité. Elle est souvent calculée de manière plus rapide (le meilleur temps de calcul est donné en grisé). Cette fonction de pénalisation a, par contre, besoin d'une première solution satisfaisant les contraintes du problème. Une solution opérationnelle possible est de faire un cycle de l'algorithme du Simplex à l'aide de la première fonction de pénalisation, puis, à partir de cette solution, lancer l'algorithme pour un grand nombre de cycles en prenant la fonction de pénalisation discontinue.

Il est très difficile de valider la qualité d'un résultat d'une allocation de ce type. Le fait d'associer la même fonction d'effort à toutes les arêtes du réseau permet de supposer que les résultats doivent être en relation avec la structure du réseau de fiabilité. Pour le vérifier, nous avons calculé à l'aide d'Aralia les facteurs d'importance qui doivent nous permettre de trier les composants en fonction de leur importance au sein du système (Pour plus d'informations sur les facteurs d'importance et leur implémentation dans un module BDD du type Aralia se référer à [DR00a]). Tous les facteurs d'importance testés renvoient le même ordre.

En dehors des permutations locales de sous-systèmes ayant des rôles très proches dans les réseaux, les allocations pour C1, C2 et P3 donnent des résultats corroborant les facteurs d'importance. Le résultat pour P4 ne semble en revanche pas conforme à nos attentes. D'autres calculs avec des jeux de paramètres différents ont été réalisés sur cette instance particulière. Leurs résultats sont validés par les facteurs d'importance.

Le Tableau 11-6 affiche l'ordre des facteurs d'importance et des 4 allocations.

Allocation				Facteur d'importance				
C1	C2	P3	P4	MIF	DIF	CIF	RAW	RRW
e17	e17	e3	e17	e17	e17	e17	e17	e17
e3	e3	e17	e9	e3	e3	e3	e3	e3
e18	e18	e18	e3	e18	e18	e18	e18	e18
e1	e1	e1	e1	e1	e1	e1	e1	e1
e9	e9	e9	e6	e9	e9	e9	e9	e9
e6	e13	e6	e18	e13	e13	e13	e13	e13
e13	e6	e13	e11	e6	e6	e6	e6	e6
e8	e8	e11	e16	e8	e8	e8	e8	e8
e11	e16	e8	e15	e11	e11	e11	e11	e11
e15	e5	e16	e8	e15	e15	e15	e15	e15
e16	e11	e15	e5	e16	e16	e16	e16	e16
e5	e15	e5	e13	e5	e5	e5	e5	e5

Tableau 11-6 : Comparaison du résultat des allocations et des facteurs d'importance pour SR93-9

11.5.3 CONCLUSION

Ce cas particulier nous montre que l'algorithme du Simplex n'est pas toujours robuste et qu'il convient de l'utiliser avec des jeux de paramètres différents afin d'éviter de rester bloqué sur un optimum local.
D'autres algorithmes permettant de prendre en compte les variables de décision continues devront sans doute être ajoutés à Aloès dans l'avenir. Ne serait ce que pour valider les résultats des algorithmes par comparaison mutuelle.

11.6 BIBLIOGRAPHIE

[SR93] S. Soh and S. Rai. *Experimental results on preprocessing of path/cut terms in sum of disjoint products technique.* IEEE Transactions on Reliability, 42(1):24–33, 1993.

[KLY99] Sy-Yen Kuo, Shyue-Kung Lu, and Fu-Min Yeh. *Determining Terminal-Pair Reliability Based on Edge Expansion Diagrams Using OBDD.* IEEE Transactions on Reliability, 48(3):234–246, September 1999.

[DR00a] Y. Dutuit and A. Rauzy. *Efficient Algorithms to Assess Components and Gates Importances in Fault Tree Analysis.* Reliability Engineering and System Safety, 72(2):213–222, 2000.

[MS91] K. Misra and U. Sharma. *An efficient Algorithm To Solve Integer-Programming Problems Arising In System-Reliability Design.* IEEE Transactions on Reliability Vol. 40 No. 1, April 1991, pp81-91

[KP00] W. Kuo and V. Prasad. *An Annoted Overview of System-Reliability Optimization.* IEEE Transactions on Reliability Vol. 49 No. 2, June 2000, pp176-187

[NM65] J.A. Nelder, R. Mead. *A simplex method for function minimisation.* Computer Journal, 1965

12. CONCLUSIONS & PERSPECTIVES

12.1 CONCLUSIONS

L'objectif principal de cette thèse était la réalisation d'outils commercialisables autour du moteur de calcul Aralia. Si ce logiciel a permis de faire un bond important dans les analyses booléennes de sûreté de fonctionnement, il lui manquait un certain nombre d'outils périphériques afin d'accroître ses potentialités.

Le premier projet, Hévéa, soutenu par les partenaires du groupe Aralia travaillant dans le domaine nucléaire, était l'étude des arbres d'événements et en particulier leur exploitation en conjonction avec les arbres de défaillance. Ce projet a permis

- d'étudier le fonctionnement usuel de ces arbres d'événements "booléens",
- de proposer un formalisme à base de diagrammes orientés acycliques bipartis (état-arête)
- d'en définir une sémantique afin de traduire ces arbres d'événements en formules booléennes, en prenant en compte le particularisme de la propagation d'événements de configuration.

Les études probabilistes de sûreté (EPS) qui ont été réalisées par les partenaires industriels ont montré la faisabilité de l'approche, mais ont poussé le logiciel Aralia dans ces derniers retranchements. Un certain nombre de séquences des arbres d'événements de ces EPS n'était pas directement quantifiable. Nous avons donc testé toutes les techniques de réécriture connues, afin de dépasser ces difficultés. Le résultat majeur de cette phase du projet est que les probabilités exactes de toutes les séquences ont pu être calculées en moins d'une heure (10 minutes pour 99% des séquences) et que certaines techniques d'approximation étaient encore plus rapides, tout en restant plus "respectueuses" que les méthodes généralement employées dans les autres logiciels de traitement des EPS.
Le calcul de la probabilité exacte des séquences d'une EPS, nous a permis d'étudier en "grandeur nature" les approximations généralement utilisées pour évaluer la sûreté des installations nucléaires internationales. Parmi les trois grandes catégories d'approximations (enveloppe monotone, utilisation de MOCUS, développement de Sylvester-Poincarré limité au premier ordre), la plus importante provient du fait que l'on ne considère pas les littéraux négatifs des implicants premiers. Cette approximation est pessimiste. Elle ne remet donc pas en cause les études de sûreté réalisées à l'aide des outils historiques d'évaluation des EPS. En revanche, des erreurs de diagnostic peuvent être engendrées à cause de cette approximation. A titre d'exemple, deux séquences dont les probabilités exactes valent respectivement 10^{-7} et 10^{-9}, peuvent toutes les deux se voir affectée une probabilité évaluée à 10^{-7}. Cela engendre forcément des actions correctives inutiles pour la seconde séquence.

Le second projet visait principalement à proposer des méthodes génériques d'allocation d'exigence et plus particulièrement d'allocation de fiabilité ou d'indisponibilité. Une méthode assez générale a été proposée pour les méthodes de type pondération. Le langage Aloès permet de décrire la grande majorité des modèles d'optimisation d'allocation de fiabilité rencontrés dans la littérature et dans la pratique industrielle. L'architecture logicielle de Aloès permet de le coupler à des moteurs de calcul extérieurs. Ainsi, des interfaces ont été mises en œuvre entre Aloès, Aralia et Réséda. Les problèmes d'allocation peuvent donc inclure directement des arbres de défaillance et des réseaux de fiabilité, en tant que modèle de description des différentes architectures du système à optimiser. D'autres moteurs de calcul extérieurs peuvent, de cette manière, être utilisés dans les description Aloès. Les performances d'Aralia, en termes de temps de calcul, ont fait que même pour des arbres de défaillance de taille importante, les temps d'évaluation probabiliste ne sont pas supérieurs aux temps d'optimisation. Les expérimentations réalisées dans le cadre de ce projet ont montré, cependant, que les algorithmes proposés dans Aloès n'étaient pas entièrement satisfaisants en termes de robustesse et de précision des résultats.

Aralia n'aurait pas connu son succès actuel sans une interface utilisateur conviviale. Pendant les quatre ans qu'a duré cette thèse, j'ai développé deux versions d'Aralia WorkShop (anciennement appelé SimTree) qui ont été commercialisées. Elles ont servi de vitrine technologique à Aralia. Une troisième version est en cours de développement et devrait être présentée à la prochaine conférence λμ (en mars 2002 à Lyon). Des demandes d'évolution, provenant tant du groupe Aralia, que des clients utilisateurs de Aralia WorkShop, nous ont poussé à étudier en particulier une loi "Test périodique" plus proche de la réalité, à intégrer des modules permettant de mettre en évidence les défaillances de cause commune existant entre événements de base et à les traiter quantitativement de manière traditionnelle (même si cela relève plus d'une approche empirique que mathématique).

12.2 PERSPECTIVES

Les perspectives à court terme concernent l'industrialisation de la version 3.0 de Aralia WorkShop. Cette version offrira une réelle interopérabilité des trois modèles booléens d'analyse des risques que sont les arbres de défaillance, les arbres d'événements et les diagrammes de fiabilité. De plus, les trois approches de calcul de Aralia (BDD, p-BDD et Mocus) seront directement disponibles au sein d'un gestionnaire de calcul évolué.

Bien qu'il soit possible d'améliorer les réécritures présentent au sein de Hévéa, cette étude n'est pas d'actualité. La plupart des difficultés rencontrées au cours de ce projet ayant été résolues, il n'y a plus de raison, autre qu'académique, de le poursuivre. Les différentes voies d'amélioration pouvant être étudiées sont décrites dans le chapitre 6 et particulièrement dans sa conclusion.

En revanche, le projet Aloès n'est pas achevé. En effet, le logiciel Aloès ne peut pas être commercialisé en l'état. Il convient dans une seconde phase de réaliser une interface graphique permettant de saisir le modèle à optimiser et de lancer le ou les processus d'optimisation à la manière d'une *calculette*. Si cette deuxième phase aboutit, il faudra nécessairement envisager une troisième phase pour utiliser Aloès dans un contexte industriel. Il conviendra alors d'améliorer les algorithmes déjà existants et d'en ajouter d'autres, comme par exemple un algorithme génétique.

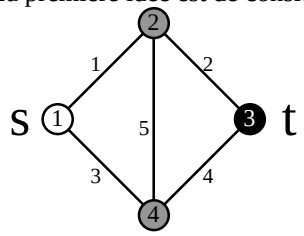
Partie IV :
Annexes

SOMMAIRE DES ANNEXES

A. ALGORITHME DE MADRE & COUDERT

L'algorithme de compilation d'un réseau de fiabilité en formule booléenne a été présenté par Madre et Coudert dans [MCFB94]. Il a été implémenté dans l'outil Réséda [DRS96]. Cette annexe est destinée à expliquer son fonctionnement.
Comme dans le cas des diagrammes de fiabilité, deux variables booléennes sont associées à chaque nœud *i* du réseau. La première, notée *panne(n_i)*, correspond à la défaillance du composant ; elle est "à vrai" lorsque le nœud *i* est défaillant, suite à une panne interne. L'autre, noté *fct(n_i)*, représente l'alimentation du nœud. Elle est "à vrai" lorsque le nœud *i* n'est pas alimenté. Il peut ne pas être défaillant, mais ne pas être alimenté si tous les composants dont il dépend ne sont pas alimentés. De la même manière, une variable booléenne , notée *panne(a_j)*, est associée à chaque arête *j* du réseau. Elle est "à vrai" lorsque l'arrête est défaillante.

La première idée est de considérer chaque arête comme une contrainte à prendre en compte.



Par exemple dans la figure ci-contre, l'arête a_5, entre le nœud n_2 et le nœud n_4, peut être lue de la manière suivante :
Si aucun des trois composants (a_5, n_2 et n_4) n'est défaillant, alors :
Si n_2 fonctionne, alors n_4 fonctionne
Et si n_4 fonctionne, alors n_2 fonctionne.

Cette contrainte associée à l'arête a_5 est notée *cst(a_5)*. Elle s'écrit alors :

$$cst(a_5) = (\neg panne(n_2) \wedge \neg panne(n_4) \wedge \neg panne(a_5)) \Rightarrow ((fct(n_2) \Rightarrow fct(n_4)) \wedge (fct(n_2) \Rightarrow fct(n_4)))$$

d'où, après simplification :

$$cst(a_5) = (\neg panne(n_2) \wedge \neg panne(n_4) \wedge \neg panne(a_5)) \Rightarrow (fct(n_2) \equiv fct(n_4))$$

Soit une arête a_j reliant deux nœuds n_i et n_k, la contrainte *cst(a_j)* est différente suivant qu'il s'agit d'une arête ou d'un arc. Une arête a_j est représentée par la contrainte :

$$cst(a_j) = (\neg panne(n_i) \wedge \neg panne(n_k) \wedge \neg panne(a_j)) \Rightarrow (fct(n_i) \equiv fct(n_k)) \qquad [13\text{-}1]$$

Lorsque l'arête a_j est unidirectionnelle (de n_i vers n_k), c'est-à-dire lorsque c'est un arc, la contrainte s'écrit :

$$cst(a_j) = (\neg panne(n_i) \wedge \neg panne(n_k) \wedge \neg panne(a_j)) \Rightarrow (fct(n_i) \Rightarrow fct(n_k)) \qquad [13\text{-}2]$$

Notons *Cst*, la contrainte globale représentant toutes les relations entre les composants du réseau.

$$Cst = \bigcap_{i=1}^{n} cst(a_i) \text{ avec } n \text{ est le nombre d'arêtes du réseau considéré.} \qquad [13\text{-}3]$$

Cette formule booléenne est construite à partir de l'ensemble de variables X = {*panne(a_1)*, ..., *panne(a_n)*, *panne(n_1)*, ..., *panne(n_m)*} et de l'ensemble de variables Y = {*fct(n_1)*, ..., *fct(n_m)*}. Les variables appartenant à Y sont nécessairement des variables intermédiaires qui devront être définies par la suite. En effet, il est clair que le fonctionnement (ou le non fonctionnement) du réseau ne dépend que des pannes pouvant survenir aux composants du réseau.

S'intéresser au fonctionnement du réseau, revient à dire :
Si la contrainte globale est satisfaite alors, si le nœud source du réseau fonctionne alors le nœud cible du réseau fonctionne également, et cela pour toutes les combinaisons des valeurs des variables *fct(x)*.
Ce qui peut également s'écrire :

$$F(X) = \forall Y\left(Cst \Rightarrow \left(fct(s) \Rightarrow fct(t)\right)\right) \qquad [13\text{-}4]$$

Cette formule booléenne fixe toutes les variables de fonctionnement. Le fonctionnement du réseau ne dépend que de l'état interne ou physique de chaque composant.

S'intéresser au dysfonctionnement du réseau revient à chercher des solutions qui satisfont la contrainte suivante :
Existe-t-il une ou plusieurs combinaisons des valeurs des variables *fct(x)* qui satisfont la contrainte suivante : $Cst \wedge fct(s) \wedge \neg fct(t)$

B. LOIS DE PROBABILITE

Le Tableau 13-1 récapitule les différentes lois de probabilité des événements de base disponibles dans Aralia WorkShop. Ces lois sont directement issues de Aralia. La seule différence réside dans le fait que l'interface vérifie la validité de la valeur des paramètres avant de les transmettre à Aralia.

Loi	Expression	Par.	Définition	Limites
Constante	$A(t) = q$	q	Probabilité de défaillance	$0 \leq$ Probabilité ≤ 1
Exponentielle	$A(t) = 1\text{-}\exp^{-\lambda.t}$	λ	Taux de défaillance	Taux ≥ 0
GLM	$A(t) = \gamma \exp^{-(\lambda+\mu).t}$ $+ \lambda / (\lambda+\mu) \cdot \exp^{-(\lambda+\mu).t}$	γ λ μ	Probabilité de refus de démarrage initial Taux de défaillance Taux de réparation	$0 \leq$ Probabilité ≤ 1 Taux ≥ 0 Taux ≥ 0
Weibull	$A(t) = 1\text{-}\exp^{-\phi(t)}$ avec $\phi(t) = ((t-\tau)/\alpha)^{\beta}$	α β τ	Paramètre d'échelle Paramètre de forme Paramètre de localisation	Facteur > 0 Facteur > 0 Temps > 0
Test Périodique	Cf. dans ce paragraphe	λ θ τ	Taux de défaillance Premier intervalle entre tests Intervalle entre deux tests consécutifs	Taux ≥ 0 Temps > 0 Temps > 0
Test Périodique complète	La méthode de calcul est présenté en détail dans [DRST99].	λ λ* μ θ τ γ π X σ ω	Taux de défaillance en cours de fonctionnement Taux de défaillance durant le test Taux de réparation (une fois la panne détectée) Premier intervalle entre tests Intervalle entre deux tests consécutifs Probabilité de défaillance due au déclenchement du test (valeur de non prise en compte : 0 (le déclenchement du test n'entraîne pas de défaillance) Durée du test Indicateur de disponibilité du composant durant le test (= 0, le composant est indisponible; = 1, il est disponible) Taux de couverture du test (probabilité que la panne du composant soit détectée lors du test) Probabilité d'oubli de reconfiguration (après le test et après la réparation)	Taux ≥ 0 Taux ≥ 0 Taux ≥ 0 Temps > 0 Temps > 0 $0 \leq$ Probabilité ≤ 1 Temps > 0 0 ou 1 $0 \leq$ Probabilité ≤ 1 $0 \leq$ Probabilité ≤ 1
NRD	$A(t) = \exp^{-\mu.d}$	μ d	Taux de réparation Délai	Taux ≥ 0 Temps > 0
Temps de mission constant	$A(t) = Q + 1 - \exp^{-\lambda.T}$	λ T Q	Taux de défaillance Durée de mission Probabilité	Taux ≥ 0 Temps > 0 $0 \leq$ Probabilité ≤ 1
GLM Asymptotique	$A(t) = \lambda / (\lambda+\mu)$	λ μ	Taux de défaillance Taux de réparation	Taux ≥ 0 Taux ≥ 0
Temps Borné	Cf. dans ce paragraphe	t_0 délai	Temps de départ Durée	Temps > 0 Temps > 0

Tableau 13-1 : Lois de probabilité des événements de base

Les développements mathématiques établissant les valeurs ou justifiant l'utilisation des quatre premières lois du Tableau 13-1 ne sont pas présentés ici. Le lecteur désireux d'étudier ces problèmes peut se reporter à l'une des monographies écrites sur le sujet [PG80, Vil88, AM93, ICM95, Coc97].

Il existe dans Aralia deux lois de test périodique. La première représente davantage un composant soumis à une politique de remplacement périodique, plutôt qu'un composant dont on teste le fonctionnement puis qu'on répare s'il est constaté défaillant. De plus, la première loi considère le remplacement comme instantané et sûr.

La probabilité d'un composant soumis à cette loi est donné par l'équation suivante :

$$A(t) = \begin{cases} 1 - \exp^{-l.t} & \text{si } t \le t \\ 1 - \exp^{-l.((t-t)\%q)} & \text{si } t > t \end{cases} \qquad [13\text{-}5]$$

avec l : le taux de défaillance du composant,
t : la date du premier test
q : la période de remplacement (intervalle de temps entre 2 remplacements consécutifs)

Les limites de cette loi nous ont poussés à définir dans [DRST99] une loi de test périodique plus réaliste et à l'intégrer au sein d'Aralia. Cette loi considère un composant pouvant tomber en panne en fonction d'un taux de défaillance. Il est testé périodiquement. Le test dure un temps donné pendant lequel le composant est soit indisponible, soit disponible en fonction d'un paramètre de la loi. Si le composant est en panne, il est alors réparé en fonction d'un taux de réparation. D'autres paramètres permettent de prendre en considération d'autres phénomènes rencontrés pour ce type de composant, tel le fait que le déclenchement du test puisse rendre défaillant le composant, que la couverture du test ne soit pas sûre ou que les personnes chargées de la maintenance oublient de reconfigurer correctement le composant à la fin du test ou de la réparation.

Les lois CMT (constant mission time) et NRD sont principalement utilisées dans le domaine nucléaire. La première permet de spécifier le comportement d'un composant n'intervenant que dans une seule phase du système. La durée de la phase est spécifiée au préalable et le composant tombe en panne en fonction d'un taux de défaillance donné. Les calculs de probabilité de l'événement-sommet considéré doivent se faire pour un temps supérieur aux temps de mission des composants. Cette hypothèse est réaliste dans le calcul d'une séquence, où on considère les temps de mission des différents systèmes de sûreté en fonction de l'inertie rencontrée au cœur des centrales nucléaires. La loi NRD, quant à elle, permet de prendre en considération la réparation d'un initiateur. Elle suppose qu'un ou plusieurs systèmes de sûreté ont permis de retarder la dégradation du système pendant un temps d. Ce délai a été mis à profit pour essayer de réparer l'initiateur. L'événement considéré répond donc à la question "Le délai d a-t-il été suffisant pour réparer l'initiateur ?".

La loi GLM Asymptotique renvoie la valeur asymptotique (pour un temps tendant vers l'infini) d'une loi GLM (Gamma Lambda Mu).

La loi temps borné est destinée à introduire la notion de temps local pour un composant. Elle comprend trois paramètres : un décalage ou temps de début d'utilisation du composant (t_0) , une durée d'utilisation (d) et enfin une loi de probabilité (*law*).
Sa valeur au temps t est définie de la façon suivante :

- Si $t < t_0$ alors c'est la valeur de la loi law au temps 0.
- Si $t_0 \le t \le t_0 + d$, alors c'est la valeur de la loi law au temps $t - t_0$.
- Finalement, si $t > t_0 + d$, alors c'est la valeur de la loi law au temps d.

Autrement dit, la loi est décalée de t_0 et tronquée après d.
Ce modificateur de loi peut être utilisé de nombreuses manières. Il généralise la loi "Temps de mission constant" (CMT) et permet de prendre en compte le cas de composants ayant déjà un certain vécu à t=0.

C. APPROXIMATIONS POUR L'EPS1

C.1 MULTIPLICATION DES PROBABILITES LE LONG D'UNE SEQUENCE

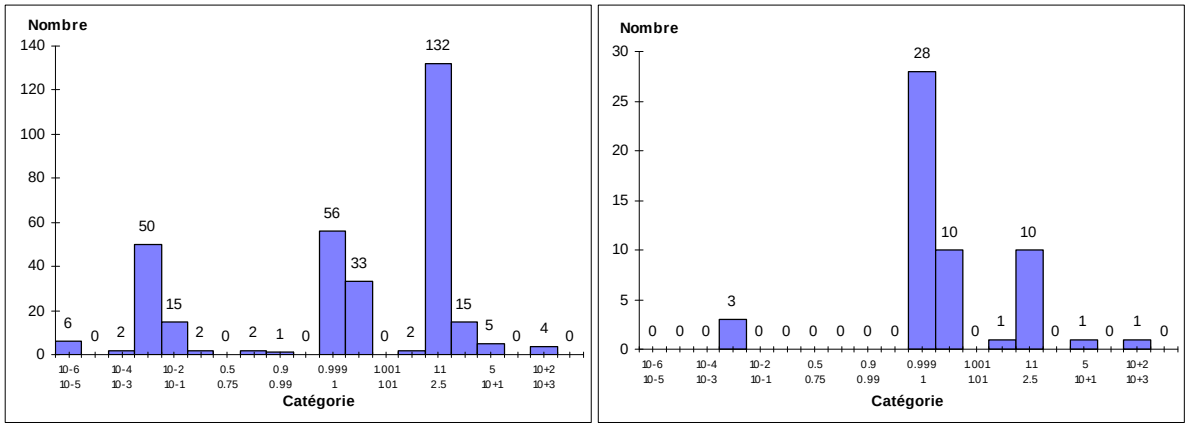


Figure 13.1 : Approximation de type "Multiplication" : Répartition du facteur d'erreur

C.2 NE PAS TENIR COMPTE DES SYSTEMES DE SURETE QUI FONCTIONNENT

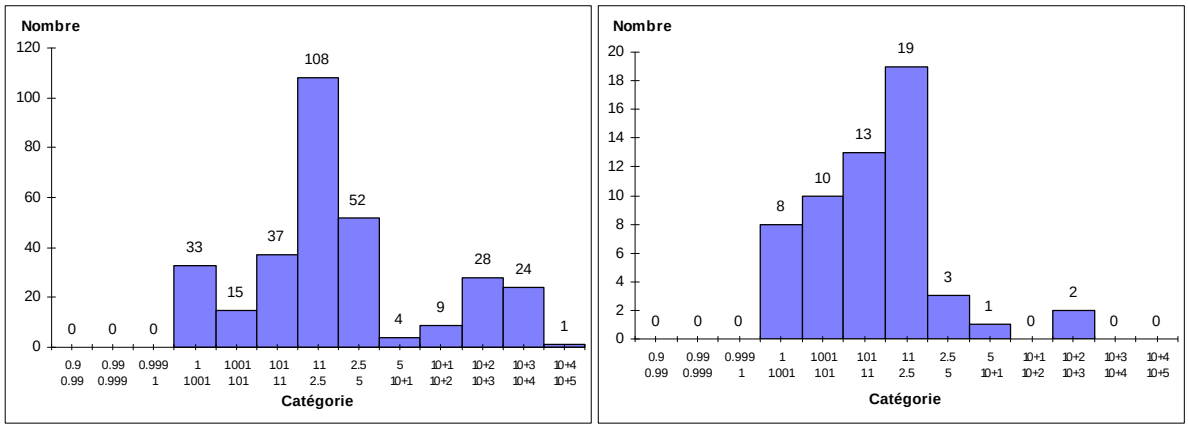


Figure 13.2 : Approximation de type "Exist-True" : Répartition du facteur d'erreur

C.3 ENVELOPPE MONOTONE

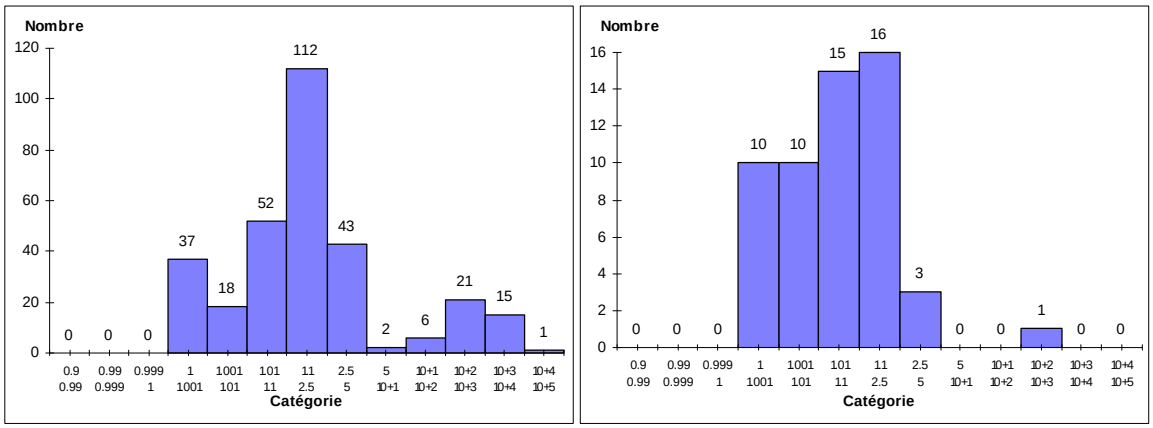


Figure 13.3 : Approximation "Enveloppe monotone" : Répartition du facteur d'erreur

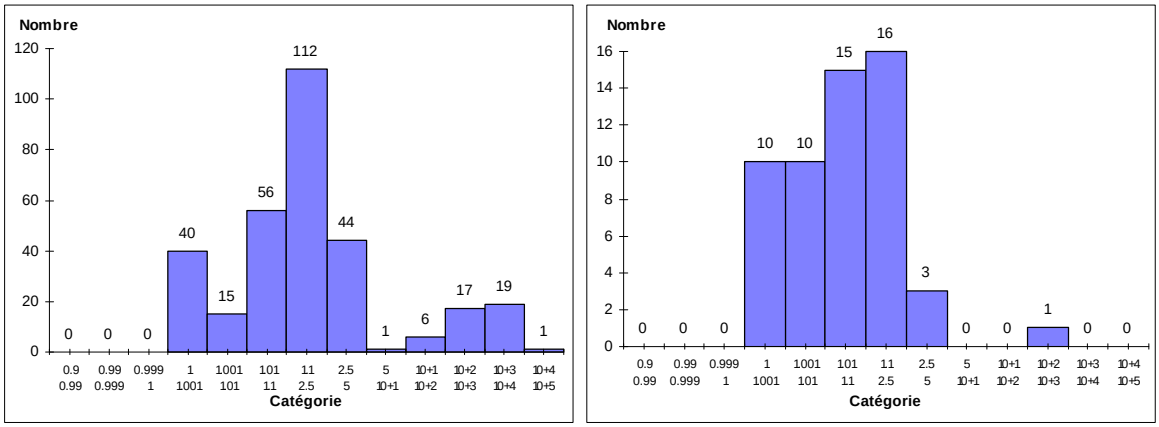


Figure 13.4 : Approximation "Quantificateur existentiel" : Répartition du facteur d'erreur

C.4 APPROXIMATION DE LA PROBABILITE

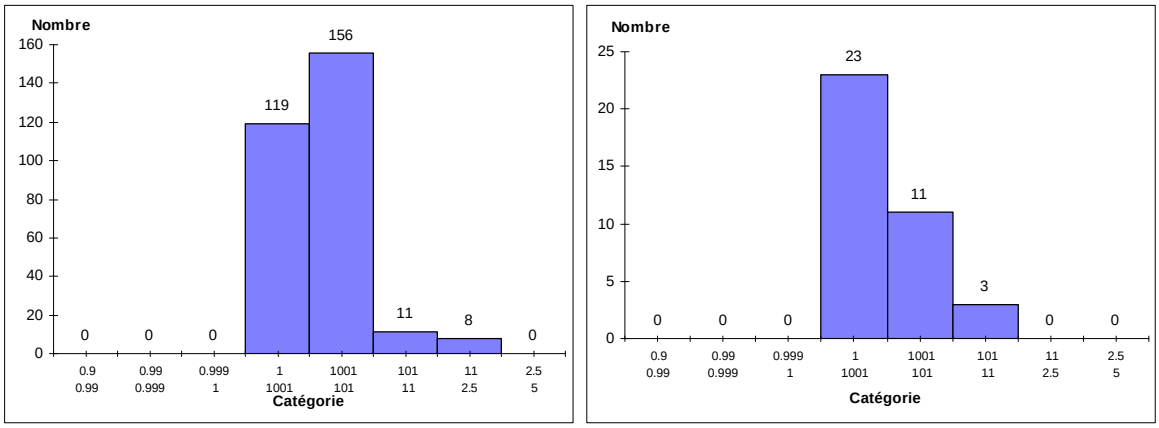


Figure 13.5 : Approximation "Somme de la probabilité des produits" : Répartition du facteur d'erreur

C.5 OBTENTION DES COUPES MINIMALES

Algorithme à la MOCUS

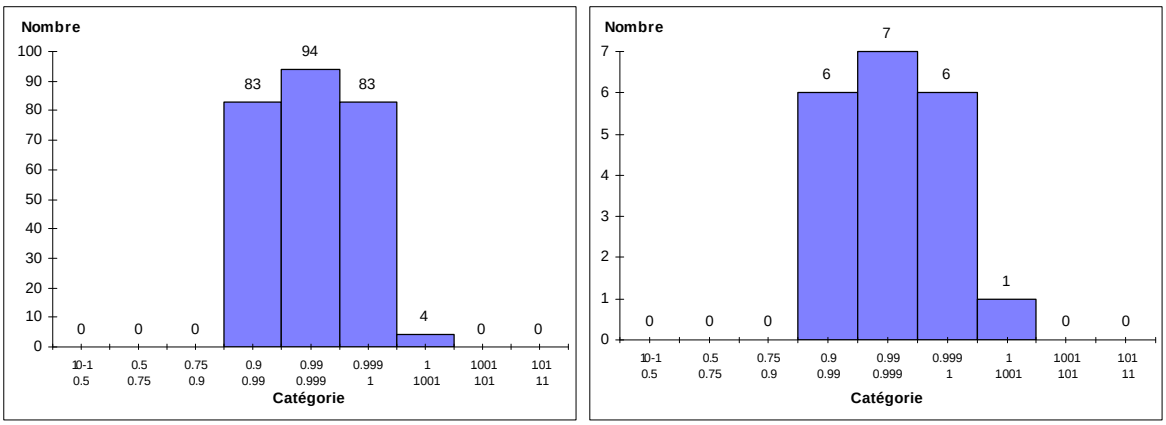


Figure 13.6 : Approximation "Mocus sur des fonctions monotones" : Répartition du facteur d'erreur

Principe du "List matching" ou des "Shadow variables"

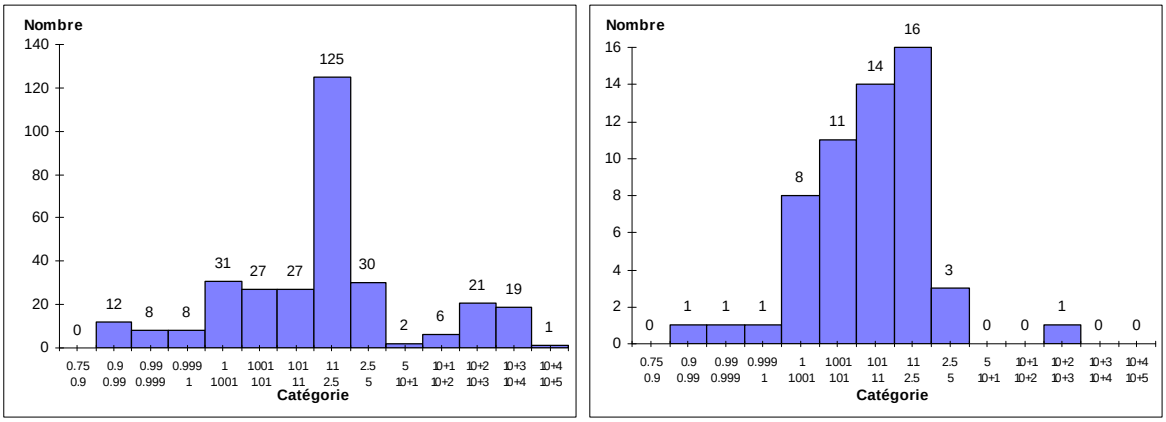


Figure 13.7 : Approximation "Mocus avec Shadow Variable" : Répartition du facteur d'erreur

C.6 APPROXIMATION "OPERATIONELLE" (GENERALEMENT EMPLOYEE)

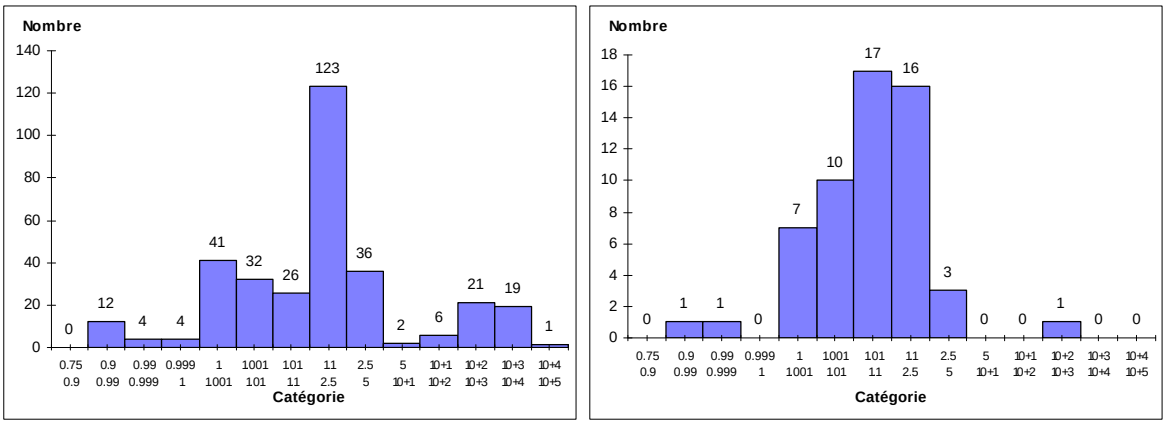


Figure 13.8 : Approximation "Opérationnelle" : Répartition du facteur d'erreur

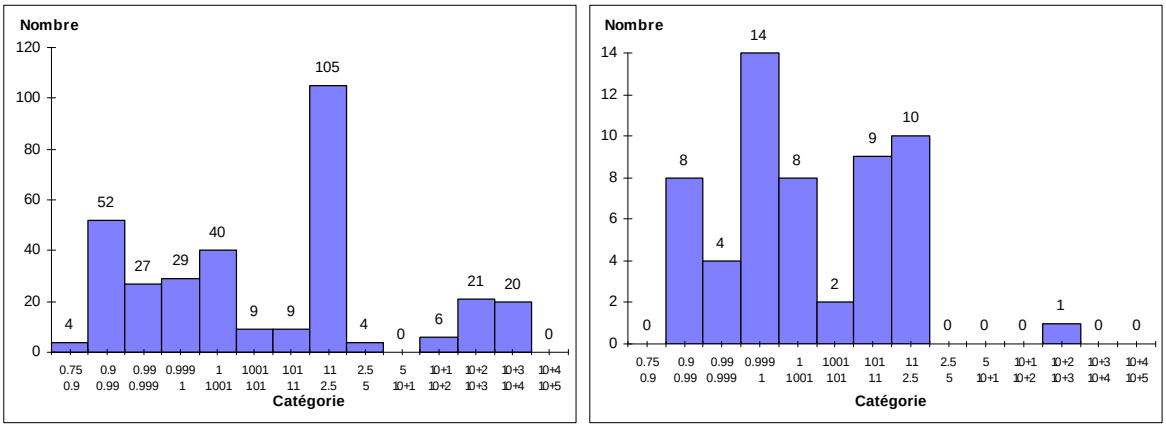


Figure 13.9 : Approximation "Opérationnelle avec facteur correctif" : Répartition du facteur d'erreur

D. HIERARCHISATION DES SEQUENCES DE L'EPS2

Le tableau ci-dessous compare la hiérarchisation (R, R+) des séquences en fonction de leurs probabilités exactes (P) et de leurs probabilités approximées (P+) lorsque l'on ne prend en compte que l'enveloppe monotone de chaque séquence. Ce tableau résulte du tri par ordre décroissant de la valeur de la probabilité approximée. Seules les 300 premières séquences ont été affichées.
La colonne Diff correspond à la différence entre R et R+ lorsque cette différence est positive.

Seq	R	R+	Diff	P	P+
260	1	1		1.00e+0	1.00e+0
269	2	2		1.00e+0	1.00e+0
278	3	3		1.00e+0	1.00e+0
285	4	4		1.00e+0	1.00e+0
303	5	5		1.00e+0	1.00e+0
245	6	6		9.70e-1	1.00e+0
250	7	7		9.70e-1	1.00e+0
255	8	8		9.70e-1	1.00e+0
292	9	9		9.45e-1	1.00e+0
299	10	10		9.45e-1	1.00e+0
323	11	11		2.97e-2	3.00e-2
300	12	12		2.92e-2	3.00e-2
293	13	13		2.92e-2	3.00e-2
246	14	14		2.91e-2	3.00e-2
251	15	15		2.91e-2	3.00e-2
256	16	16		2.91e-2	3.00e-2
301	17	17		2.60e-2	2.60e-2
296	18	18		2.60e-2	2.60e-2
199	19	19		6.56e-3	6.60e-3
472	20	20		4.04e-3	4.20e-3
501	21	21		3.37e-3	3.40e-3
1370	22	22		1.89e-3	1.90e-3
142	23	23		1.40e-3	1.40e-3
1801	42	24	18	2.42e-4	9.76e-4
257	24	25		8.95e-4	8.95e-4
247	25	26		8.94e-4	8.95e-4
252	26	27		8.94e-4	8.95e-4
1720	27	28		8.80e-4	8.85e-4
159	28	29		5.99e-4	6.00e-4
1682	29	30		5.35e-4	5.39e-4
982	30	31		4.69e-4	5.00e-4
1808	32	32		4.47e-4	4.88e-4
1802	43	33	10	2.24e-4	4.88e-4
543	31	34		4.56e-4	4.60e-4
658	33	35		4.18e-4	4.40e-4
324	34	36		3.00e-4	3.00e-4
279	35	37		3.00e-4	3.00e-4
286	36	38		3.00e-4	3.00e-4
261	37	39		3.00e-4	3.00e-4
270	38	40		3.00e-4	3.00e-4
304	39	41		2.91e-4	3.00e-4
191	40	42		2.80e-4	2.80e-4
937	41	43		2.48e-4	2.50e-4
1775	52	44	8	5.76e-5	2.30e-4
1700	44	45		1.45e-4	1.49e-4

Seq	R	R+	Diff	P	P+
1397	45	46		1.19e-4	1.30e-4
320	56	47	9	3.81e-5	1.27e-4
1788	46	48		1.15e-4	1.15e-4
1776	53	49	4	5.74e-5	1.15e-4
306	47	50		1.03e-4	1.08e-4
485	49	51		9.91e-5	1.00e-4
1032	48	52		9.94e-5	1.00e-4
322	50	53		8.89e-5	8.89e-5
957	51	54		6.94e-5	7.40e-5
175	54	55		4.79e-5	4.80e-5
1432	55	56		4.56e-5	4.60e-5
586	58	57	1	3.65e-5	3.80e-5
623	59	58	1	3.65e-5	3.80e-5
1734	57	59		3.70e-5	3.73e-5
1764	60	60		3.18e-5	3.46e-5
316	68	61	7	1.60e-5	3.36e-5
210	63	62	1	2.71e-5	3.30e-5
1811	61	63		3.11e-5	3.26e-5
1805	69	64	5	1.56e-5	3.26e-5
1006	62	65		2.76e-5	2.76e-5
1466	74	66	8	1.14e-5	2.70e-5
483	64	67		2.37e-5	2.52e-5
1007	65	68		2.16e-5	2.30e-5
693	66	69		2.09e-5	2.20e-5
498	67	70		2.03e-5	2.10e-5
317	73	71	2	1.18e-5	1.76e-5
473	70	72		1.52e-5	1.68e-5
520	71	73		1.50e-5	1.56e-5
878	72	74		1.31e-5	1.40e-5
1467	75	75		1.07e-5	1.35e-5
507	76	76		1.07e-5	1.15e-5
671	78	77	1	8.50e-6	1.05e-5
1385	79	78	1	8.40e-6	9.50e-6
305	77	79		9.00e-6	9.00e-6
1400	81	80	1	7.01e-6	8.62e-6
1594	80	81		7.81e-6	8.23e-6
1747	88	82	6	5.61e-6	8.16e-6
228	84	83	1	6.62e-6	8.03e-6
669	82	84		6.99e-6	7.92e-6
1814	86	85	1	6.09e-6	7.78e-6
502	83	86		6.73e-6	7.23e-6
775	85	87		6.56e-6	7.00e-6
318	87	88		5.87e-6	5.87e-6
394	94	89	5	4.35e-6	5.60e-6
313	105	90	15	2.40e-6	5.28e-6

Seq	R	R+	Diff	P	P+
1574	90	91		4.77e-6	5.02e-6
1809	93	92	1	4.50e-6	4.88e-6
1803	106	93	13	2.25e-6	4.88e-6
1732	89	94		4.78e-6	4.78e-6
212	91	95		4.62e-6	4.76e-6
433	95	96		4.32e-6	4.70e-6
89	92	97		4.58e-6	4.61e-6
307	126	98	28	1.28e-6	4.40e-6
1755	114	99	15	1.92e-6	4.11e-6
981	96	100		4.08e-6	4.08e-6
1371	97	101		3.72e-6	4.04e-6
1671	99	102		3.18e-6	3.46e-6
1630	98	103		3.18e-6	3.23e-6
308	109	104	5	2.08e-6	3.12e-6
369	102	105		2.62e-6	3.00e-6
1699	100	106		2.92e-6	2.92e-6
314	115	107	8	1.92e-6	2.88e-6
51	101	108		2.77e-6	2.80e-6
1272	103	109		2.58e-6	2.80e-6
1477	110	110		2.06e-6	2.70e-6
1718	104	111		2.54e-6	2.54e-6
1000	108	112		2.12e-6	2.50e-6
1570	180	113	67	4.38e-7	2.37e-6
1766	107	114		2.20e-6	2.31e-6
691	112	115		2.00e-6	2.20e-6
564	111	116		2.03e-6	2.11e-6
1082	117	117		1.88e-6	2.00e-6
1604	113	118		1.94e-6	1.97e-6
675	119	119		1.76e-6	1.92e-6
833	116	120		1.89e-6	1.90e-6
295	118	121		1.86e-6	1.90e-6
1571	123	122	1	1.55e-6	1.85e-6
1683	133	123	10	1.02e-6	1.81e-6
659	122	124		1.57e-6	1.76e-6
154	120	125		1.69e-6	1.72e-6
298	121	126		1.65e-6	1.65e-6
551	149	127	22	7.66e-7	1.56e-6
359	124	128		1.52e-6	1.52e-6
1475	134	129	5	1.00e-6	1.46e-6
1584	125	130		1.32e-6	1.39e-6
1031	127	131		1.27e-6	1.27e-6
953	130	132		1.12e-6	1.25e-6
853	129	133		1.13e-6	1.20e-6
422	131	134		1.05e-6	1.20e-6
230	128	135		1.13e-6	1.16e-6
1399	171	136	35	5.56e-7	1.14e-6
309	132	137		1.04e-6	1.04e-6
474	136	138		9.73e-7	1.01e-6
1558	137	139		9.59e-7	1.01e-6
1512	142	140	2	8.45e-7	1.01e-6
984	203	141	62	2.91e-7	1.00e-6
1819	135	142		9.76e-7	9.76e-7
730	139	143		9.52e-7	9.60e-7
315	138	144		9.58e-7	9.58e-7
349	143	145		8.37e-7	9.26e-7
1420	148	146	2	7.68e-7	9.10e-7

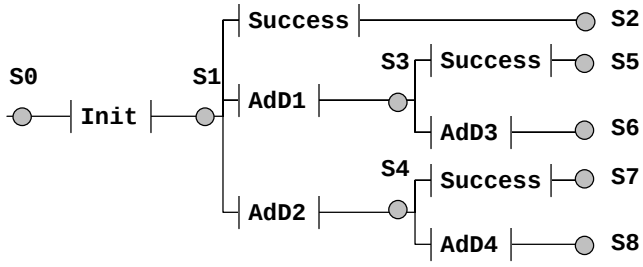
Seq	R	R+	Diff	P	P+
599	141	147		8.97e-7	9.09e-7
636	179	148	31	4.48e-7	9.09e-7
1813	140	149		9.00e-7	9.07e-7
1807	178	150	28	4.50e-7	9.07e-7
310	162	151	11	6.40e-7	9.00e-7
380	156	152	4	6.75e-7	8.70e-7
1401	155	153	2	7.01e-7	8.62e-7
903	145	154		7.78e-7	8.30e-7
1468	165	155	10	6.23e-7	8.16e-7
1654	170	156	14	5.60e-7	8.16e-7
1717	144	157		7.93e-7	8.07e-7
490	151	158		7.44e-7	8.01e-7
1816	153	159		7.09e-7	8.00e-7
484	147	160		7.71e-7	7.94e-7
552	152	161		7.19e-7	7.80e-7
1411	158	162		6.50e-7	7.80e-7
69	150	163		7.56e-7	7.75e-7
902	146	164		7.72e-7	7.72e-7
670	154	165		7.04e-7	7.25e-7
508	161	166		6.42e-7	6.96e-7
985	169	167	2	5.71e-7	6.90e-7
395	166	168		5.93e-7	6.72e-7
499	157	169		6.61e-7	6.61e-7
200	167	170		5.81e-7	6.60e-7
1406	159	171		6.49e-7	6.59e-7
160	160	172		6.47e-7	6.51e-7
1386	172	173		5.35e-7	6.48e-7
407	163	174		6.30e-7	6.30e-7
1478	164	175		6.26e-7	6.29e-7
1492	174	176		5.16e-7	6.17e-7
750	168	177		5.72e-7	6.10e-7
1544	351	178	173	3.03e-8	6.01e-7
213	189	179	10	3.67e-7	5.94e-7
990	312	180	132	4.85e-8	5.82e-7
1545	176	181		4.75e-7	5.65e-7
1617	173	182		5.35e-7	5.44e-7
1751	188	183	5	3.82e-7	5.44e-7
706	181	184		4.26e-7	5.26e-7
1036	184	185		4.20e-7	5.00e-7
1701	204	186	18	2.78e-7	5.00e-7
939	249	187	62	1.39e-7	5.00e-7
991	209	188	21	2.42e-7	4.95e-7
1396	175	189		4.75e-7	4.75e-7
27	177	190		4.53e-7	4.61e-7
637	183	191		4.21e-7	4.54e-7
503	182	192		4.23e-7	4.39e-7
928	185	193		4.18e-7	4.20e-7
800	186	194		3.94e-7	4.20e-7
1698	214	195	19	2.27e-7	4.14e-7
1662	233	196	37	1.92e-7	4.11e-7
704	192	197		3.49e-7	3.96e-7
1794	223	198	25	2.04e-7	3.87e-7
1782	262	199	63	1.02e-7	3.87e-7
799	187	200		3.86e-7	3.86e-7
1572	418	201	217	1.52e-8	3.79e-7
533	193	202		3.47e-7	3.74e-7

Seq	R	R+	Diff	P	P+
975	199	203		3.14e-7	3.70e-7
1595	252	204	48	1.32e-7	3.68e-7
1573	190	205		3.64e-7	3.64e-7
370	191	206		3.58e-7	3.60e-7
940	197	207		3.22e-7	3.60e-7
1800	208	208		2.47e-7	3.55e-7
1476	200	209		3.04e-7	3.39e-7
262	194	210		3.33e-7	3.37e-7
271	195	211		3.33e-7	3.37e-7
1731	196	212		3.30e-7	3.32e-7
211	206	213		2.74e-7	3.30e-7
223	229	214	15	1.97e-7	3.30e-7
1812	198	215		3.14e-7	3.26e-7
1806	242	216	26	1.57e-7	3.26e-7
1040	201	217		3.01e-7	3.20e-7
434	207	218		2.51e-7	3.12e-7
192	202	219		3.00e-7	3.04e-7
1765	244	220	24	1.49e-7	3.04e-7
944	366	221	145	2.57e-8	2.91e-7
1	205	222		2.75e-7	2.80e-7
1769	220	223		2.16e-7	2.76e-7
1407	211	224		2.34e-7	2.70e-7
1372	218	225		2.18e-7	2.68e-7
311	234	226	8	1.74e-7	2.60e-7
354	213	227		2.31e-7	2.56e-7
1532	221	228		2.14e-7	2.56e-7
1002	210	229		2.36e-7	2.50e-7
408	231	230	1	1.94e-7	2.50e-7
945	253	231	22	1.31e-7	2.47e-7
1596	212	232		2.34e-7	2.36e-7
1673	217	233		2.24e-7	2.35e-7
597	226	234		2.01e-7	2.28e-7
634	227	235		2.01e-7	2.28e-7
477	219	236		2.17e-7	2.26e-7
249	215	237		2.25e-7	2.25e-7
254	216	238		2.25e-7	2.25e-7
1575	279	239	40	8.04e-8	2.25e-7
1452	261	240	21	1.04e-7	2.11e-7
825	222	241		2.09e-7	2.10e-7
1697	224	242		2.01e-7	2.02e-7
1746	225	243		2.01e-7	2.01e-7
692	228	244		2.01e-7	2.01e-7
221	230	245		1.94e-7	1.96e-7
103	232	246		1.92e-7	1.94e-7
621	235	247		1.73e-7	1.90e-7
656	236	248		1.73e-7	1.90e-7
1273	256	249	7	1.18e-7	1.86e-7
511	238	250		1.69e-7	1.83e-7
1060	237	251		1.69e-7	1.80e-7
547	239	252		1.67e-7	1.73e-7
1196	240	253		1.60e-7	1.70e-7
1502	248	254		1.42e-7	1.70e-7
1548	241	255		1.60e-7	1.68e-7
1513	423	256	167	1.43e-8	1.60e-7
1439	293	257	36	6.61e-8	1.56e-7
339	243	258		1.52e-7	1.52e-7

Seq	R	R+	Diff	P	P+
587	284	259	25	7.32e-8	1.52e-7
624	285	260	25	7.32e-8	1.52e-7
959	332	261	71	3.89e-8	1.48e-7
544	251	262		1.37e-7	1.47e-7
1514	245	263		1.45e-7	1.46e-7
231	271	264	7	8.94e-8	1.45e-7
1576	246	265		1.43e-7	1.44e-7
423	247	266		1.43e-7	1.44e-7
116	250	267		1.37e-7	1.39e-7
1642	254	268		1.25e-7	1.36e-7
1752	310	269	41	4.91e-8	1.33e-7
955	255	270		1.25e-7	1.25e-7
1479	299	271	28	6.05e-8	1.24e-7
170	257	272		1.18e-7	1.20e-7
1104	258	273		1.13e-7	1.20e-7
676	260	274		1.06e-7	1.16e-7
1034	289	275	14	6.85e-8	1.16e-7
1025	270	276		9.12e-8	1.15e-7
1716	298	277	21	6.16e-8	1.14e-7
728	263	278		9.99e-8	1.10e-7
226	259	279		1.09e-7	1.09e-7
960	274	280		8.81e-8	1.07e-7
660	266	281		9.49e-8	1.06e-7
1453	267	282		9.33e-8	1.06e-7
381	268	283		9.21e-8	1.04e-7
1405	307	284	23	5.43e-8	1.03e-7
393	264	285		9.78e-8	9.78e-8
1493	455	286	169	8.75e-9	9.78e-8
712	321	287	34	4.50e-8	9.62e-8
1546	686	288	398	9.62e-10	9.62e-8
1547	265	289		9.52e-8	9.52e-8
531	276	290		8.65e-8	9.38e-8
143	269	291		9.20e-8	9.27e-8
329	278	292		8.37e-8	9.26e-8
1089	408	293	115	1.70e-8	9.26e-8
1494	272	294		8.85e-8	8.90e-8
353	273	295		8.85e-8	8.85e-8
694	325	296	29	4.19e-8	8.78e-8
312	275	297		8.67e-8	8.67e-8
1402	280	298		7.79e-8	8.62e-8
965	451	299	152	9.10e-9	8.61e-8
1431	277	300		8.53e-8	8.53e-8

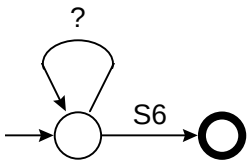
E. AUTOMATES DE THOMPSON

La manière la plus efficace pour déterminer si un chemin (ou une séquence) de l'arbre d'événements est reconnu par le langage de description de séquences dérivant d'expressions régulières est de transformer l'expression régulière en un automate qui reconnaît les mêmes chemins. Pour simplifier, utilisons une analogie : un chemin peut être vu comme une chaîne de caractères dont les lettres correspondent aux états parcourus par ce chemin.



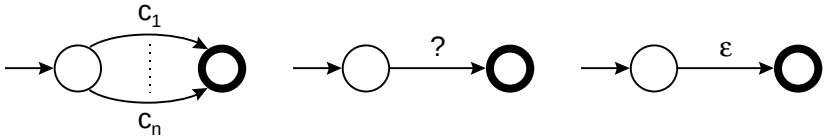
Déterminer si le chemin reliant les états S0 et S6 est reconnu par l'expression régulière ?*.{S6} est équivalent à déterminer si cette expression régulière reconnaît la chaîne de caractères S0.S1.S3.S6.

Intuitivement, un automate est une machine simplifiée. Il est composé d'un certain nombre d'états (représentés par des cercles) reliés entre eux par des transitions (représentées par des flèches). Les transitions sont étiquetées par une lettre, qui correspond à un état de l'arbre d'événement, par un joker noté ? représentant n'importe quel état ou par le mot vide ε.



La règle du jeu est la suivante : à partir d'un état, on a le droit de passer dans un autre état, soit parce qu'il existe une transition marquée ε de l'état de départ vers l'état d'arrivée, soit parce qu'il existe une transition marquée x (où x est le prochain caractère de la chaîne à reconnaître) reliant l'état de départ à l'état d'arrivée. L'automate ci-dessus reconnaît tous les mots qui finissent pas S6.
Dans un automate, certains états sont marqués états terminaux (en trait plus épais). Un des états est marqué comme état initial (signalé par une petite flèche entrante). Le jeu consiste à essayer de trouver un chemin qui part de l'état initial et aboutit à un état terminal, après avoir lu tous les caractères de la chaîne donnée en entrée. Si un tel chemin existe, on dit que l'automate reconnaît la chaîne.

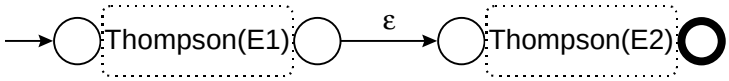
L'algorithme de "construction de Thompson" transforme une expression régulière en un automate qui a la particularité d'avoir un seul état terminal (appelé aussi état final par symétrie avec l'état initial de l'automate). De plus, aucune transition ne sort de cet état final.
La construction de Thompson procède par récurrence sur la structure de l'expression régulière. Pour les trois expressions régulières atomiques (respectivement un ensemble de lettres définies (c_1, ... c_n), une lettre quelconque et le mot vide ε), les automates construits sont les suivants :



L'automate de gauche reconnaît uniquement les chaînes mono-caractères c_1, ..., c_n. celui du milieu toutes les chaînes constituées d'un seul caractère et celui de droite uniquement le mot vide.

Pour l'expression régulière E1.E2 (qui correspond à la mise en séquence des deux expressions régulières E1 et E2), on construit les automates de Thompson correspondant à E1 et E2, puis on met une transition étiquetée ε de l'état final de l'automate de E1 vers l'état initial de l'automate de E2.

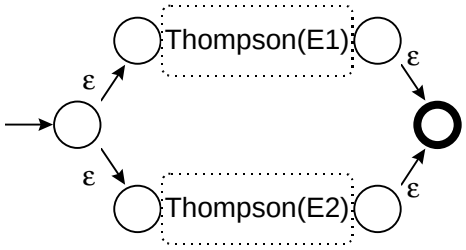
E1.E2



L'automate résultant reconnaît les concaténations d'un mot reconnu par Thompson(E1) et d'un mot reconnu par Thompson(E2).
En suivant un raisonnement semblable, on construit les automates associés aux opérateurs :

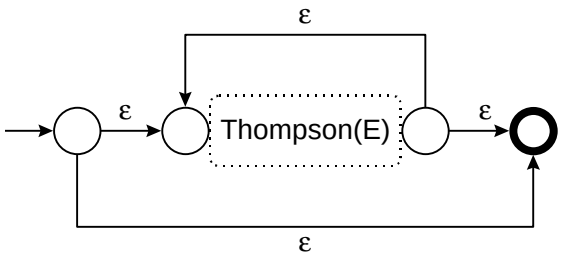
- d'union (ou d'alternative)

E1 | E2



- et de répétition.

E*



Cette présentation des automates de Thompson est très largement inspirée de [WL93].

[WL93] P. Weis et X. Leroy. *Chapitre 15 du "Le langage CAML" : Rechercher de motif dans un texte.* InterEdition. ISBN 2-7296-0639-4. 1993.

F. PROBLEMES AU LANGAGE ALOES POUR SR93-5

F.1 REDONDANCE ACTIVE SANS DCC

```
require reseda ;

model Syst {

  reseda {{
    network SR93-5;     /* Déclaration du réseau */
      nodes             /* Liste des nœuds du réseau */
        Source, N2, N3, N4, N5, N6, N7, N8, Target;
      labels            /* Liste des arêtes du réseau */
        A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12;
      edges             /* Description du réseau */
        Source <- A1 -> N2,
        Source <- A2 -> N3,
        N2 <- A3 -> N5,
        N3 <- A4 -> N4,
        N3 <- A5 -> N7,
        N4 <- A6 -> N5,
        N5 <- A7 -> N6,
        N5 <- A8 -> N8,
        N6 <- A9 -> N7,
        N7 <- A10 -> N8,
        N7 <- A11 -> Target,
        N8 <- A12 -> Target;
      source Source;    /* Definition de la source */
      target Target;    /* Definition de la cible */
      reliable          /* Composants fiables */
        Source, N2, N3, N4, N5, N6, N7, N8, Target;
    ;

    compile SR93-5;     /* Directive de compilation */
    ranking BFS SR93-5; /* Indexation pour le DBD */
    print store SR93-5; /* Fonction booléenne */
  }} ;

  /* Declaration variables de decision */
  decision int choice.A1 = [1, 3];
  decision int choice.A2 = [1, 3];
  decision int choice.A3 = [1, 3];
  decision int choice.A4 = [1, 3];
  decision int choice.A5 = [1, 3];
  decision int choice.A6 = [1, 3];
  decision int choice.A7 = [1, 3];
  decision int choice.A8 = [1, 3];
  decision int choice.A9 = [1, 3];
  decision int choice.A10 = [1, 3];
  decision int choice.A11 = [1, 3];
  decision int choice.A12 = [1, 3];

  /* Declaration des constantes */
  float cstB = 10;
  float prbB = 0.01;

  /* Definition des modificateurs de module externe */
  reseda("law", "A1", "constant", pow(prbB, choice.A1);
  reseda("law", "A2", "constant", pow(prbB, choice.A2);
  reseda("law", "A3", "constant", pow(prbB, choice.A3);
  reseda("law", "A4", "constant", pow(prbB, choice.A4);
  reseda("law", "A5", "constant", pow(prbB, choice.A5);
  reseda("law", "A6", "constant", pow(prbB, choice.A6);
  reseda("law", "A7", "constant", pow(prbB, choice.A7);
  reseda("law", "A8", "constant", pow(prbB, choice.A8);
  reseda("law", "A9", "constant", pow(prbB, choice.A9);
  reseda("law", "A10", "constant", pow(prbB, choice.A10);
  reseda("law", "A11", "constant", pow(prbB, choice.A11);
  reseda("law", "A12", "constant", pow(prbB, choice.A12);

  /* Definition des variables sommets */
  float Proba = reseda("Pr", 0);
  float Cost = (choice.A1 + choice.A2 + choice.A3
             + choice.A4 + choice.A5 + choice.A6
             + choice.A7 + choice.A8 + choice.A9
             + choice.A10 + choice.A11 + choice.A12
              ) * cstB;

  /* Declaration des dependances */
  dependenty Cost * + ;
  dependenty Proba * - ;
};

problem probl1.Syst of Syst {
  optimize < Cost ;
  subject-to
    Proba <= 3.0006e-008;
};

...

problem probl3.Syst of Syst {
  optimize < Proba ;
  subject-to
    Cost <= 240;
};
```

F.2 REDONDANCE AVEC DCC

```
require reseda ;

model Syst {

  reseda {{
    network SR93-5;      /* Déclaration du réseau */
      ... /* Idem modèle précédent précédent */
  }} ;

  /* Declaration variables de decision */
  ... /* Idem modèle précédent précédent */

  /* Declaration des constantes */
  float cstB = 10;
  float prbB = 0.01;
  float beta2 = 0.03;
  float beta3 = 0.01;

  /* Declaration des variables intermédiaires */
  float beta.A1 = switch choice.A1 {1:0, 2:beta2, 3:beta3};
  float proba.A1 = (pow(((1 - beta.A1) * prbB), choice.A1)
                 + (beta.A1 * prbB));
  float beta.A2 = switch choice.A2 {1:0, 2:beta2, 3:beta3};
  float proba.A2 = (pow(((1 - beta.A2) * prbB), choice.A2)
                 + (beta.A2 * prbB));
  float beta.A3 = switch choice.A3 {1:0, 2:beta2, 3:beta3};
  float proba.A3 = (pow(((1 - beta.A3) * prbB), choice.A3)
                 + (beta.A3 * prbB));
  float beta.A4 = switch choice.A4 {1:0, 2:beta2, 3:beta3};
  float proba.A4 = (pow(((1 - beta.A4) * prbB), choice.A4)
                 + (beta.A4 * prbB));
  float beta.A5 = switch choice.A5 {1:0, 2:beta2, 3:beta3};
  float proba.A5 = (pow(((1 - beta.A5) * prbB), choice.A5)
                 + (beta.A5 * prbB));
  float beta.A6 = switch choice.A6 {1:0, 2:beta2, 3:beta3};
  float proba.A6 = (pow(((1 - beta.A6) * prbB), choice.A6)
                 + (beta.A6 * prbB));
  float beta.A7 = switch choice.A7 {1:0, 2:beta2, 3:beta3};
  float proba.A7 = (pow(((1 - beta.A7) * prbB), choice.A7)
                 + (beta.A7 * prbB));
```

```
  float beta.A8 = switch choice.A8 {1:0, 2:beta2, 3:beta3};
  float proba.A8 = (pow(((1 - beta.A8) * prbB), choice.A8)
                + (beta.A8 * prbB));
  float beta.A9 = switch choice.A9 {1:0, 2:beta2, 3:beta3};
  float proba.A9 = (pow(((1 - beta.A9) * prbB), choice.A9)
                + (beta.A9 * prbB));
  float beta.A10 = switch choice.A10 {1:0, 2:beta2, 3:beta3};
  float proba.A10 = (pow(((1 - beta.A10) * prbB), choice.A10)
                + (beta.A10 * prbB));
  float beta.A11 = switch choice.A11 {1:0, 2:beta2, 3:beta3};
  float proba.A11 = (pow(((1 - beta.A11) * prbB), choice.A11)
                + (beta.A11 * prbB));
  float beta.A12 = switch choice.A12 {1:0, 2:beta2, 3:beta3};
  float proba.A12 = (pow(((1 - beta.A12) * prbB), choice.A12)
                + (beta.A12 * prbB));

  /* Definition des modificateurs de module externe */
  reseda("law", "A1", "constant", proba.A1);
  reseda("law", "A2", "constant", proba.A2);
  reseda("law", "A3", "constant", proba.A3);
  reseda("law", "A4", "constant", proba.A4);
  reseda("law", "A5", "constant", proba.A5);
  reseda("law", "A7", "constant", proba.A7);
  reseda("law", "A6", "constant", proba.A6);
  reseda("law", "A8", "constant", proba.A8);
  reseda("law", "A9", "constant", proba.A9);
  reseda("law", "A10", "constant", proba.A10);
  reseda("law", "A11", "constant", proba.A11);
  reseda("law", "A12", "constant", proba.A12);

  /* Definition des variables sommets */
  float Proba = reseda("Pr", 0);
  float Cost = (choice.A1 + choice.A2 + choice.A3
             + choice.A4 + choice.A5 + choice.A6
             + choice.A7 + choice.A8 + choice.A9
             + choice.A10 + choice.A11 + choice.A12
              ) * cstB;

  /* Declaration des dependances */
  dependenty Proba * - ;
  dependenty Cost * + ;
};

problem probl1.Syst of Syst {
  optimize < Cost ;
  subject-to
    Proba <= 4.66288e-007;
};

...

problem probl3.Syst of Syst {
  optimize < Proba ;
  subject-to
    Cost <= 240;
};
```

F.3 TEST PERIODIQUE

```
require reseda ;

model Syst {

  reseda {{
    network SR93-5;       /* Déclaration du réseau */
     ... /* Idem modèle précédent précédent */
  }} ;

  /* Declaration variables de decision */
  decision int choice.A1 = [1, 4];
  decision int choice.A2 = [1, 4];
  decision int choice.A3 = [1, 4];
  decision int choice.A4 = [1, 4];
  decision int choice.A5 = [1, 4];
  decision int choice.A6 = [1, 4];
  decision int choice.A7 = [1, 4];
  decision int choice.A8 = [1, 4];
  decision int choice.A9 = [1, 4];
  decision int choice.A10 = [1, 4];
  decision int choice.A11 = [1, 4];
  decision int choice.A12 = [1, 4];

  /* Declaration des constantes */
  float cstB = 10;
  float lbd = 0.01;
  float T1 = 10;
  float T2 = 20;
  float T3 = 40;
  float T4 = 80;
  float TM = 500;
  float step = 2.5;
  float Cih = 50;

  /* Declaration des variables intermediaires */
  float period.A1 = switch choice.A1 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A1 = (int((TM / period.A1)) * cstB);
  float period.A2 = switch choice.A2 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A2 = (int((TM / period.A2)) * cstB);
  float period.A3 = switch choice.A3 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A3 = (int((TM / period.A3)) * cstB);
  float period.A4 = switch choice.A4 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A4 = (int((TM / period.A4)) * cstB);
  float period.A5 = switch choice.A5 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A5 = (int((TM / period.A5)) * cstB);
  float period.A6 = switch choice.A6 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A6 = (int((TM / period.A6)) * cstB);
  float period.A7 = switch choice.A7 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A7 = (int((TM / period.A7)) * cstB);
  float period.A8 = switch choice.A8 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A8 = (int((TM / period.A8)) * cstB);
  float period.A9 =
        switch choice.A9 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A9 = (int((TM / period.A9)) * cstB);
  float period.A10 =
        switch choice.A10 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A10 = (int((TM / period.A10)) * cstB);
  float period.A11 =
        switch choice.A11 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A11 = (int((TM / period.A11)) * cstB);
  float period.A12 =
        switch choice.A12 {1:T1, 2:T2, 3:T3, 4:T4};
  float cost.A12 = (int((TM / period.A12)) * cstB);

  /* Definition des modificateurs de module externe */
  reseda("law", "A1", "periodic-test", lbd, period.A1, 0);
  reseda("law", "A2", "periodic-test", lbd, period.A2, 0);
  reseda("law", "A3", "periodic-test", lbd, period.A3, 0);
  reseda("law", "A4", "periodic-test", lbd, period.A4, 0);
  reseda("law", "A5", "periodic-test", lbd, period.A5, 0);
  reseda("law", "A6", "periodic-test", lbd, period.A6, 0);
  reseda("law", "A7", "periodic-test", lbd, period.A7, 0);
  reseda("law", "A8", "periodic-test", lbd, period.A8, 0);
  reseda("law", "A9", "periodic-test", lbd, period.A9, 0);
  reseda("law", "A10", "periodic-test", lbd, period.A10, 0);
  reseda("law", "A11", "periodic-test", lbd, period.A11, 0);
  reseda("law", "A12", "periodic-test", lbd, period.A12, 0);

  /* Definition des variables sommets */
  float IndispoMoy = reseda("MPr", 0, TM, step);
  float MPCost = cost.A1 + cost.A2 + cost.A3
               + cost.A4 + cost.A5 + cost.A6
               + cost.A7 + cost.A8 + cost.A9
               + cost.A10 + cost.A11 + cost.A12 ;
  float Cost = MPCost + (IndispoMoy * TM * Cih);

  /* Declaration des dependances */
  dependenty Cost * ? ;
  dependenty IndispoMoy * + ;
};

problem probl1.Syst of Syst {
  optimize < Cost ;
  subject-to
    IndispoMoy <= 0.0494539;
};

...

problem probl3.Syst of Syst {
  optimize < IndispoMoy ;
  subject-to
    Cost <= 4236.35;
};
```

F.4 TRUELOVE

```
require reseda ;

model Syst {

  reseda {{
    network SR93-5;      /* Déclaration du réseau */
      ... /* Idem modèle précédent précédent */
  }} ;

  /* Declaration variables de decision */
  decision float choice.A01 = [0, 1];
  decision float choice.A02 = [0, 1];
  decision float choice.A03 = [0, 1];
  decision float choice.A04 = [0, 1];
  decision float choice.A05 = [0, 1];
  decision float choice.A06 = [0, 1];
  decision float choice.A07 = [0, 1];
  decision float choice.A08 = [0, 1];
  decision float choice.A09 = [0, 1];
  decision float choice.A10 = [0, 1];
  decision float choice.A11 = [0, 1];
  decision float choice.A12 = [0, 1];

  /* Declaration des constantes */
  float QRex = 0.01;
  float CRex = 30;
  float a = QRex * CRex;

  /* Definition des variables intermediaires */
  float Cost = a/choice.A01 + a/choice.A02 + a/choice.A03
             + a/choice.A04 + a/choice.A05 + a/choice.A06
             + a/choice.A07 + a/choice.A08 + a/choice.A09
             + a/choice.A10 + a/choice.A11 + a/choice.A12
             ;
```

```
  /* Definition des modificateurs de module externe */
  reseda("law", "A01", "constant", choice.A01);
  reseda("law", "A02", "constant", choice.A02);
  reseda("law", "A03", "constant", choice.A03);
  reseda("law", "A04", "constant", choice.A04);
  reseda("law", "A05", "constant", choice.A05);
  reseda("law", "A06", "constant", choice.A06);
  reseda("law", "A07", "constant", choice.A07);
  reseda("law", "A08", "constant", choice.A08);
  reseda("law", "A09", "constant", choice.A09);
  reseda("law", "A10", "constant", choice.A10);
  reseda("law", "A11", "constant", choice.A11);
  reseda("law", "A12", "constant", choice.A12);

  float Proba = reseda("Pr", 0);
};

problem probl1.Syst of Syst {
  optimize < Cost ;
  subject-to
    Proba <= 0.000305937;
};

...

problem probl3.Syst of Syst {
  optimize < Proba ;
  subject-to
    Cost <= 360;
};
```

SOMMAIRE DES TABLEAUX

SOMMAIRE DES FIGURES

Cette thèse synthétise les travaux conduits et les résultats obtenus dans le cadre de deux projets de recherche soutenus par des partenaires industriels et décrits ci-après:

- Le projet **Hévéa** visait à traduire les séquences des arbres d'événements (AdE) en formules booléennes. La première phase du projet a abouti à une formalisation du problème et à la réalisation d'un logiciel permettant cette traduction. La deuxième phase a permis de trouver des solutions permettant de diminuer les problèmes d'explosion combinatoire du traitement par diagrammes binaires de décision (BDD : Binary Decision Diagram) des formules booléennes ainsi générées. La troisième phase a validé les approximations généralement utilisées lors des Etudes Probabilistes de Sûreté (EPS) des centrales nucléaires au terme d'une étude systématique portant sur deux EPS industrielles.
- Le projet **Aloès** avait pour but d'étudier des problèmes d'allocation d'exigences s'appuyant sur des modèles booléens d'analyse des risques. Après une recherche bibliographique et des discussions avec les partenaires du projet, il a abouti à la création d'un langage générique de description de problèmes d'allocation. Ce langage, ainsi que le logiciel **Aloès** supportant le langage, sont suffisamment génériques pour être couplé à des moteurs des calculs externes comme **Aralia** (voir ci-dessous) ou **Réséda** (outil traduisant un réseau de fiabilité en formule booléenne). Le logiciel offre alors trois algorithmes d'optimisation (descente rapide, Branch & Bound, Simplex) afin de résoudre les problèmes d'allocation. Des expérimentations sur un ensemble de réseaux de fiabilité ont permis de valider le pouvoir d'expression du langage, mais ont conclu sur la nécessité d'améliorer les algorithmes d'optimisation intégrés.

L'ensemble de ces travaux a débouché sur la création de l'atelier logiciel **Aralia WorkShop** de saisie et de traitement des modèles booléens d'analyse des risques (Arbre de défaillance : AdD, Arbre d'événements : AdE, Bloc diagramme de fiabilité : BDF), s'appuyant pour la partie traitement sur le cœur de calcul **Aralia**. Ce dernier se base sur un codage des formules booléennes en BDD.

Mots-Clés :
Atelier SdF, Arbre d'événements, Arbre de défaillance, Réseau de fiabilité, Défaillance de Cause Commune, Loi test périodique, BDD, Diagramme Binaire de Décision, Formule booléenne, Réordonnancement, Réécriture, Allocation d'exigence, Branch & Bound, Optimisation

This thesis presents the results obtained by the author during his participation to two research programs supported by industrial partners:

- The **Hévéa** projet
  The main objective of this project was to assess event-tree sequences by means of Boolean techniques. During the first stage of this program, the author formalized the problem. He designed the Hévéa software which is dedicated to event-tree assessment. In the second stage, the author introduced several original means to reduce the exponential blow-up of the BDD-based processing of Boolean formulae. During the third stage, he validated by means of a systematic study of two industrial PSA (Probabilistic Safety Analysis) the approximations commonly used in the nuclear framework.
- The **Aloes** project
  The aim of this project was the study of so-called requirement allocation problems. This project began by an important bibliographic work and ended by the creation of a language suitable for allocation problems. This language, and related software Aloès are generic enough to be connected to external assessment softwares such as **Aralia** (which encodes boolean formulae by using BDD) and **Réséda** (which transform a reliability network into boolean formulae). Three optimisation algorithms (steepest descent, branch and bound, simplex) are implemented in the **Aloès** Software to solve allocation problems. Several experiments on reliability networks validated language. These experiments showed that some of previous embedded optimisation algorithms must be improved.

The final result of all these studies is the software set **Aralia WorkShop** which is devoted to the assessment of Boolean risk analysis models (Faut-tree, Event-tree, Reliability block-diagrams and networks) thanks to the embedded computing software **Aralia**.

Keywords :
RAMS workshop, Event tree, Fault tree, Reliability network, Common cause failure, Periodic test law, BDD, Binary Decision Diagram, Boolean formula, Reordering, Rewriting, Requirement allocation, Branch & Bound, Optimisation