UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# UN MODÈLE INTELLIGENT D'ESTIMATION DES COÛTS DE DÉVELOPPEMENT DE LOGICIELS

THÈSE
PRÉSENTÉE
COMME EXIGENCE PARTIELLE
DU DOCTORAT EN INFORMATIQUE COGNITIVE

PAR
ALI IDRI

SEPTEMBRE 2003

TABLE DES MATIÈRES

# Avant-PROPOS

Je tiens à remercier très vivement la personne qui a su me motiver à entreprendre cette recherche et m'aider de ses précieux conseils; il m'importe en effet d'exprimer toute ma gratitude et ma reconnaissance à Alain Abran, mon Directeur de recherche et professeur à l'École de Technologie Supérieure de Montréal. Ses directives, son ouverture d'esprit, son soutien scientifique et son enthousiasme pour mon projet de recherche m'ont énormément aidé pour réaliser ce travail.

Je tiens à remercier aussi chaleureusement Serge Robert, mon codirecteur de recherche et professeur au département de philosophie de l'Université du Québec À Montréal. Ses observations et ses remarques m'ont amené souvent à remettre en question la pertinence de mes idées, lesquelles sont souvent améliorées en intégrant ses suggestions.

Je tiens à remercier les membres du jury : Houari Sahraoui de l'Université de Montréal, Pierre Poirier et Mounir Boukadoum de l'Université du Québec À Montréal pour leurs commentaires et leurs suggestions.

Je tiens à remercier Taghi Khoshgoftaar, professeur et Directeur du *Empirical Software Engineering Laboratory* à *Florida Atlantic University*, pour l'intérêt qu'il a porté à mon travail et pour ses remarques pertinentes.

Je tiens à remercier Ghislain Levesque, Directeur du programme de Doctorat en Informatique Cognitive, ainsi que tout le personnel administratif de ce programme de toute l'aide logistique qu'ils m'ont apporté durant mes études à l'Université du Québec À Montréal.

Une thèse est une entreprise de longue haleine, souvent passionnante mais parfois aussi source de moments de découragement. C'est dans de tels instants que l'on apprécie d'autant plus le soutien de sa famille et de ses amis. Je voudrai leur dire ici à tous ma gratitude à tous: mes deux sœurs Nadia et Amina, mes deux frères Abdelillah et Mohamed, mes amis à Montréal Ali et Sophie, Youness, Asma et Kais, Hamdan, Adel, et Hicham, mes amis au Maroc Samir, Belkasmi, Elkoutbi, Azziz, Romadi, Berbia et tous les autres. Merci à tous.

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# LISTE DES ABRÉVIATIONS

| | |
|---|---|
| AKDSI | Adjusted Kilo Delivered Source Instructions |
| ANGEL | ANAloGy Estimation tool |
| ANN | Artifical Neural Networks |
| CBR | Case-Based Reasoning |
| CTP | Computational Theory of Perception |
| CW | Computing with Words |
| FP | Function Points |
| F_ANGEL | *Fuzzy Analogy* Estimation tooL |
| FSS | Features Subset Selection |
| IA | Intelligence Artificielle |
| KISL | Kilo Instructions Sources Livrées |
| KDSI | Kilo Delivered Source Instructions |
| MIQ | Machine Intelligent Quotient |
| MMRE | Mean Magnitude Relative Error |
| MRE | Magnitude Relative Error |
| MSE | Mean Square Error |
| NC | Normal Condition |
| Ln | Logarithme népérien |
| Pred | Prediction level |
| RBFN | Radial Basis Function Networks |
| RIM | Regular Increasing Monotone Quantifier |

# RÉSUMÉ

L'objectif principal de cette thèse est le développement et la validation d'un modèle dit *intelligent* d'estimation des coûts de développement de logiciels. Notre modèle d'estimation adopte un raisonnement par analogie auquel nous avons intégré la logique floue et les réseaux de neurones afin de satisfaire nos trois critères d'intelligence : 1) la tolérance aux imprécisions tout au long du processus d'estimation, 2) la gestion des incertitudes au niveau des estimations fournies, et 3) l'apprentissage.

Le processus d'estimation de notre modèle, nommé *Fuzzy Analogy*, est composé de trois étapes. La première étape consiste en la description des projets logiciels par un ensemble d'attributs considérés comme étant les principaux conducteurs du coût. Ces attributs peuvent être évalués par des valeurs numériques ou linguistiques. Dans le cas de valeurs linguistiques, *Fuzzy Analogy* utilise une représentation floue afin de tolérer les imprécisions au niveau de la description des projets logiciels. La deuxième étape porte sur l'évaluation de la similarité entre le nouveau projet, pour lequel on veut estimer le coût, et tous les projets logiciels historiques. Pour ce faire, nous avons développé un ensemble de mesures de similarité entre les projets logiciels. Ces mesures opèrent en deux niveaux: 1) similarité individuelle et 2) similarité globale. Les mesures de similarité individuelle se basent sur des techniques d'agrégation floue telles que les méthodes *max-min* et *sum-product*. Les mesures de similarité globale, quant à elles, utilisent des quantificateurs linguistiques monotones croissants, tels que *most*, *many* et *all*, pour combiner les différentes valeurs des similarités individuelles entre deux projets logiciels. La troisième étape du processus d'estimation de *Fuzzy Analogy* consiste à utiliser les coûts réels des projets historiques *étroitement similaires* au nouveau projet pour en déduire une estimation à son coût.

Nous avons validé notre modèle en utilisant les 63 projets du COCOMO'81. Dans un premier temps, nous avons évalué la performance de *Fuzzy Analogy* en termes de deux critères: la précision des estimations et la tolérances des imprécisions tout au long du processus d'estimation. Nous avons comparé la performance de notre modèle avec celles de trois autres techniques d'estimation. Les résultats obtenus avantagent notre modèle sur les trois autres techniques. Dans une deuxième étape, nous avons validé la procédure de gestion des incertitudes au niveau des estimations fournies par *Fuzzy Analogy*. Cette procédure se base sur la version non-déterministe de l'affirmation *similar software projects have similar costs*. Ainsi, notre modèle permet de générer un ensemble de valeurs estimées, avec une fonction de distribution des possibilités, du coût d'un nouveau projet. Cette fonction de distribution associe à chaque valeur estimée un nombre de l'intervalle (0,1) représentant le degré de possibilité pour que cette valeur soit la valeur réelle du coût du nouveau projet. La fonction de distribution peut être aussi utilisée pour la gestion des risques dus aux incertitudes dans les estimations. La dernière étape de validation consiste à évaluer la procédure d'apprentissage adoptée par notre modèle. Cette procédure d'apprentissage a pour objectif la mise à jour des différents paramètres

de notre modèle afin de répondre aux nouveaux besoins de l'environnement concerné. Dans un premier temps, nous avons doté le prototype logiciel F_ANGEL de toutes les fonctionnalités permettant à l'estimateur la mise à jour des paramètres de *Fuzzy Analogy*. Cette stratégie de mise à jour nécessite donc une expertise de la part de l'estimateur. Afin de faciliter cette tâche à l'estimateur, nous avons proposé une stratégie qui permette de retrouver automatiquement les valeurs adéquates de certains paramètres de *Fuzzy Analogy*, spécifiquement ceux relatifs aux valeurs linguistiques décrivant les projets logiciels. Cette stratégie utilise un réseau de neurones RBFN (Radial Basis Function Networks) équivalent à notre modèle. Ainsi, ces paramètres sont déterminés par le réseau RBFN et utilisés, ensuite, par *Fuzzy Analogy*.

Le cadre de recherche que nous avons choisi pour cette thèse est relativement large. Ainsi de nombreux travaux ou programmes de recherche peuvent être envisagés. Nos perspectives ne concernent donc pas seulement le domaine d'estimation des coûts; elles touchent aussi d'autres domaines tels que la logique floue, la mesure de logiciels, l'intelligence artificielle et le raisonnement par analogie.

**Mots-clés** : Estimation des coûts de logiciels, Génie logiciel, Logique floue, Raisonnement par analogie, Réseau de neurones RBFN, Intelligence artificielle.

# INTRODUCTION

## 1. CADRE DE LA THÈSE

L'estimation des coûts de développement de logiciels est une tâche très complexe dans la gestion d'un projet logiciel. En effet, sans cette estimation, les gestionnaires et les autres responsables d'un projet logiciel ne peuvent assigner les budgets, affecter les ressources nécessaires et/ou établir les plans adéquats pour mener à bien le projet logiciel. La croissance de ces coûts ainsi que les difficultés que l'on rencontre pour les prévoir et les contrôler constituent, encore aujourd'hui, une préoccupation des chercheurs et des responsables logiciels. C'est donc pour rationaliser le processus d'estimation du coût d'un projet logiciel que plusieurs techniques d'estimation ont été proposées (Boehm, 1981). Les techniques se basant sur la modélisation sont les plus citées et les mieux documentées dans la littérature. Elles peuvent être regroupées en deux catégories (Shepperd et Schofield, 1997):

- Les modèles paramétriques, et
- les modèles non-paramétriques.

Les modèles paramétriques se basent essentiellement sur une équation centrale exprimant l'effort en fonction d'un certain nombre d'attributs considérés comme étant les principaux conducteurs du coût (*cost drivers*). Ils sont mis au point en utilisant des méthodes statistiques (régression linéaire, analyse bayesienne, etc.) ou des méthodes d'analyse numérique telle que l'interpolation linéaire. Des exemples de ces modèles sont le modèle d'Halstead (Halstead, 1975), IBM-FSD (Walston et Felix, 1977) PUTNAM-SLIM (Putnam, 1978), COCOMO'81 (Boehm, 1981), COCOMO II (Boehm, 1995), et ceux se basant sur les points de fonctions (Albrecht et Gaffney, 1983; Abran et Robillard, 1994; Matson, Barrett et Mellichamp, 1994).

Les modèles non-paramétriques, quant à eux, modélisent la relation exprimant l'effort en fonction des conducteurs du coût en utilisant des techniques d'intelligence artificielle telles

que le raisonnement par analogie, les réseaux de neurones, les algorithmes génétiques, les systèmes à base de règles et les arbres de décision. Ils ont été développés pour pallier certains inconvénients des modèles paramétriques, spécifiquement celui relatif à la forme de la relation exprimant l'effort en fonction des conducteurs des coûts. En effet, les modèles non-paramétriques n'exigent, au préalable, aucune forme spécifique à cette relation. Des exemples de modèles non-paramétriques d'estimation des coûts sont publiés dans (Vicinanza et Prietulla, 1990; Porter et Selby, 1990a; Porter et Selby, 1990b; Bisio et Malabocchia, 1995; Srinivasan et Fisher, 1995; Serluca, 1995; Hughes, 1996; Shepperd et Schofield, 1997; Wittig et Finnie, 1997; Kadoda et al., 2000; Dolado, 2000; Burgess et Lefley, 2001).

Chaque type de modèle d'estimation a des avantages et des inconvénients et aucun n'a pu prouver qu'il performe mieux que les autres, en termes de précision des estimations, dans toutes les situations (Shepperd et Schofield, 1997; Niessink et Van Vliet, 1997; Briand, Langely et Wieczorek, 2000; Myrtveit et Stensrud, 1999). En effet, la performance d'un modèle dépend, en plus de l'approche adoptée pour sa mise au point, de plusieurs caractéristiques de la base de projets historiques telles que sa taille, le nombre d'attributs décrivant les projets logiciels et la présence des points extrêmes (Shepperd et Kadoda, 2001).

Le raisonnement par analogie ou le raisonnement à base de cas (*Case-Based Reasoning* – CBR) est une technique prometteuse qui s'adapte bien au problème d'estimation des coûts. En effet, elle peut être utilisée lorsqu'on a peu de connaissances et d'informations sur le problème à résoudre et pour lequel une solution optimale est a priori inconnue. Le raisonnement par analogie consiste en l'utilisation d'un historique de données afin de fournir une solution au problème étudié (Shank, 1982; Gentner, 1983; Aha, 1991; Aamodt et Plaza, 1994; Kolodner, 1993; Leake, 1996). Il a été utilisé avec succès dans plusieurs sous-disciplines du génie logiciel telles que la prédiction des erreurs dans un logiciel, la prédiction de la qualité logicielle, la conception et la réutilisation des composants logiciels (Bartsch-Spoerl, 1995; Keddar et Bareiss, 1994; Ganesan, Khoshgoftaar et Allen, 2000; Khoshgoftaar, Cukic et Seliya, 2002; Ramamoorthy, Chandra et Oshihara, 1992). Récemment, l'utilisation du raisonnement par analogie en estimation des coûts a été l'objet de plusieurs travaux de recherche (Vicinanza et Prietulla, 1990; Shepperd et Schofield, 1997; Angelis et Stamelos,

2000; Kadoda et al., 2000). En effet, le raisonnement par analogie présente plusieurs caractéristiques désirables sur les autres techniques d'estimation:

- Il est facile à expliquer aux utilisateurs;
- il utilise la connaissance dans sa forme brute;
- il peut modéliser les relations complexes existantes entre les différents attributs d'un logiciel; et
- il peut généraliser plusieurs types de modélisation en intelligence artificielle.

En général, le processus de raisonnement par analogie le plus adopté en estimation des coûts est composé de trois étapes (Shepperd et Schofield, 1997): 1) Identification des projets logiciels, y compris le nouveau projet pour lequel on veut estimer le coût, par un nombre de variables significatives et indépendantes; 2) Évaluation de la similarité entre le nouveau projet logiciel et tous les projets logiciels historiques; 3) Utilisation des valeurs des coûts réels des projets logiciels *similaires* au nouveau projet pour en déduire une estimation à son coût. Cependant, jusqu'à présent, l'utilisation du raisonnement par analogie en estimation des coûts présente trois limitations majeures: premièrement, il ne traite pas convenablement le cas des projets logiciels décrits par des valeurs linguistiques telles que *bas* et *élevé*; deuxièmement, il ne gère pas les incertitudes au niveau des estimations fournies; troisièmement, il ne permet pas un apprentissage automatique.

## 2. PROBLÉMATIQUES ET OBJECTIFS

Le raisonnement par analogie trouve son originalité dans le fait que souvent les humains l'utilisent pour résoudre et pour manipuler des situations très complexes dans leurs vies quotidiennes (Gentner, 1983; Gentner, 1998; Gentner, Holyoak et Kokinov, 2001; Holyoak et Taghard, 1991). Cependant, le raisonnement par analogie chez les humains est souvent approximatif plutôt qu'exact. En effet, le cerveau humain est capable de manipuler des informations imprécises, vagues, incertaines et partielles. Aussi, le cerveau humain est capable d'apprendre et d'opérer dans un contexte où la gestion de l'incertitude est indispensable. Dans cette thèse, nous proposons un modèle d'estimation des coûts dit *intelligent*, qui s'inspire des caractéristiques du raisonnement par analogie chez les humains.

Il doit donc tolérer les imprécisions tout au long de son processus d'estimation, gérer les incertitudes et permettre l'apprentissage afin de pourvoir traiter les situations complexes rencontrées dans la prédiction du coût en fonction d'autres facteurs de logiciels. Notre modèle intelligent d'estimation sera référé dans ce document par *Fuzzy Analogy*.

## 2.1 Tolérance des imprécisions

L'importance de la tolérance des imprécisions tout au long du processus d'estimation par analogie est due au fait que les projets logiciels sont souvent décrits par des valeurs linguistiques plutôt que par des valeurs numériques. Par exemple, dans le cas du modèle COCOMO'81, 16 parmi 17 facteurs sont mesurés sur une échelle composée de six valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé*, *très élevé* et *extra-élevé* (Boehm, 1981). L'utilisation des valeurs linguistiques par un modèle d'estimation nécessite donc la prise en considération des imprécisions et des incertitudes souvent rattachées à ce type de valeurs. Cependant, la plupart des modèles d'estimation existants, y compris l'estimation par analogie, utilisent encore la logique classique (deux valeurs de vérité: *vraie* ou *fausse*) pour représenter et pour traiter ces valeurs linguistiques. Or, une représentation classique des valeurs linguistiques ne permet pas un traitement convenable des imprécisions car elle ne ressemble pas, a priori, à celle utilisée par le cerveau humain (Zadeh, 2001). Afin de remédier à cette limitation, nous utilisons la logique floue pour la représentation et pour le traitement des valeurs linguistiques dans *Fuzzy Analogy*. En effet, la logique floue représente ces valeurs linguistiques par des ensembles flous plutôt que par des ensembles classiques (Zadeh, 1965). Un ensemble flou a l'avantage d'avoir une fonction d'appartenance dont les valeurs de vérité sont situées dans l'intervalle [0,1]. Par conséquent, il permet de modéliser convenablement la partialité dans l'appartenance à la valeur linguistique associée. Au niveau de l'évaluation de la similarité entre les projets logiciels, nous développons un ensemble de nouvelles mesures de similarité qui peuvent être utilisées au cas où les projets logiciels seraient décrits par des valeurs linguistiques. Ces mesures de similarité utilisent des techniques d'agrégation floue et des quantificateurs linguistiques monotones croissants.

## 2.2 Gestion des incertitudes

La gestion des incertitudes, au niveau des estimations fournies par notre modèle, signifie qu'elle doit générer un ensemble de valeurs estimées, plutôt qu'une seule, avec une distribution de possibilités au coût d'un projet logiciel. Cette fonction de distribution indique les degrés de certitude associés aux différentes valeurs possibles au coût du projet logiciel. La nécessité de la gestion des incertitudes au niveau des estimations de notre modèle est justifiée par:

- Une estimation est une évaluation d'une situation future d'une entité particulière (Kitchenham et Linkman, 1997).

- Les gestionnaires préfèrent, en général, avoir plusieurs valeurs estimées du coût, plutôt qu'une seule. Ceci leur offrira plus de flexibilité au niveau de la gestion des projets logiciels.

- L'estimation par analogie est basée sur l'affirmation *Similar software projects have similar costs*. Cette affirmation présente deux sources d'incertitude: tout d'abord sa conséquence est imprécise, ensuite, elle peut être non-déterministe, c'est-à-dire, qu'on peut avoir deux projets logiciels similaires, selon une mesure de similarité prédéterminée, mais leurs coûts sont totalement différents.

Nous utilisons la théorie de possibilité pour permettre à *Fuzzy Analogy* de gérer les incertitudes au niveau de ses estimations. Ainsi, *Fuzzy Analogy* fournira un ensemble de valeurs estimées, avec une distribution de possibilités, au coût d'un nouveau projet. La théorie de possibilité est une sous-discipline de la logique floue; elle a été initiée par Zadeh en 1979 et développée davantage par Dubois et Prade (Zadeh, 1979; Dubois et Prade, 1991). Nous adoptons, en particulier, l'approche de Dubois et al. pour modéliser la version non-déterministe de l'affirmation de base du raisonnement par analogie (Dubois et al., 1999). Ainsi, notre modèle adopte l'affirmation non-déterministe *Similar software projects have possibly similar costs* pour déduire toutes les valeurs possibles au coût d'un nouveau projet logiciel. L'intégration de la gestion des incertitudes dans *Fuzzy Analogy* concerne seulement la dernière étape de son processus d'estimation: étape d'Adaptation.

## 2.3 Apprentissage

Un modèle intelligent d'estimation doit être capable de s'adapter aux nouveaux changements survenus dans son environnement. La qualité d'adaptation permettra à notre modèle de modifier les valeurs de ses paramètres et ceci, afin d'ajuster ses estimations des coûts et d'être en cohérence avec son environnement. L'apprentissage est une stratégie adéquate qui pourra permettre à un modèle d'estimation de s'adapter continuellement aux changements d'état de son environnement. La nécessité d'intégration de l'apprentissage dans notre modèle d'estimation provient du fait que l'industrie des logiciels est continuellement évolutive. En effet,

- On ne développe plus le même type d'application. Aujourd'hui, l'emphase est mise sur les applications distribuées et les applications Web alors que, dans les décades précédentes, elle était mise sur les applications mono-poste classiques.

- La complexité des applications informatiques actuelles est de plus en plus élevée, et cela est due à la diversité des besoins accrus en matière d'informatisation de la plupart des services dans nos sociétés modernes.

- Le personnel impliqué dans le développement et/ou la maintenance de logiciels est de plus en plus qualifié.

- On utilise de nouvelles méthodes de développement et/ou de maintenance.

La stratégie que nous adoptons pour intégrer l'apprentissage dans *Fuzzy Analogy* consiste essentiellement en la mise à jour de ses différents paramètres tels que les définitions des différentes valeurs linguistiques (nombre de valeurs, fonctions d'appartenance associées, pondérations associées, etc.), la définition du quantificateur linguistique utilisé pour évaluer la similarité entre deux projets logiciels, et le nombre de projets similaires à considérer pour déduire une estimation au coût d'un nouveau projet logiciel. Dans un premier temps, nous intégrons dans le logiciel prototype de *Fuzzy Analogy*, F_ANGEL, des fonctionnalités qui permettront à l'estimateur de modifier tous les paramètres de *Fuzzy Analogy*. Vu que cette solution nécessite une expertise de la part de l'estimateur, nous développons une nouvelle

stratégie pour la mise à jour automatique de certains paramètres de *Fuzzy Analogy*, spécifiquement ceux relatifs aux valeurs linguistiques. Cette stratégie se base sur l'établissement d'une équivalence entre *Fuzzy Analogy* et un type particulier de réseaux de neurones: les réseaux RBFN (*Radial Basis Function Networks*). Ainsi, l'estimateur pourra utiliser le réseau RBFN, équivalent à *Fuzzy Analogy*, pour retrouver les valeurs adéquates de tous les paramètres relatifs aux valeurs linguistiques utilisées dans la description des projets logiciels.

## 3. STRUCTURE DE LA THÈSE

Le présent document est composé de cinq chapitres. Dans le premier chapitre, nous présentons les différentes techniques d'estimation des coûts de développement de logiciels. L'emphase est mise particulièrement sur la technique d'estimation basée sur la modélisation. Cette technique a permis la mise au point de ce qu'on appelle communément en littérature *les modèles d'estimation des coûts*. Ces modèles peuvent être regroupés en deux types: les modèles paramétriques et les modèles non-paramétriques. Ainsi, pour chaque type de modèle, nous présentons les concepts de base, les avantages, les inconvénients et des exemples. Nous terminons ce chapitre par une synthèse sur les travaux de recherche ayant comme objectif la validation et la comparaison des performances de ces deux types de modèle sur des bases de projets logiciels.

Le deuxième chapitre est consacré à la présentation de la plupart des concepts et des paradigmes que nous utiliserons pour intégrer les trois caractéristiques d'intelligence dans *Fuzzy Analogy*. Ainsi, dans un premier temps, nous présentons le rôle de la logique floue dans le développement des systèmes intelligents. Nous discutons, en particulier, des avantages de l'utilisation de la logique floue pour la représentation et le traitement des imprécisions et des incertitudes dans les systèmes intelligents. Ensuite, nous présentons les concepts de base de la logique floue: Ensemble flou, Proposition floue, Règle floue, Implication floue et Agrégation floue. Ce sont ces concepts que nous utiliserons pour la tolérance des imprécisions tout au long du processus d'estimation de *Fuzzy Analogy* ainsi que pour la gestion des incertitudes au niveau de ses estimations de coûts.

Le troisième chapitre traite de l'utilisation massive des valeurs linguistiques par la plupart des modèles d'estimation des coûts. Ces modèles adoptent une représentation classique de ces valeurs linguistiques. Ainsi, nous présentons les inconvénients d'une telle représentation et nous suggérons l'utilisation d'une représentation floue. Pour illustrer les avantages d'une représentation floue, nous appliquons la logique floue pour la représentation et pour le traitement des six valeurs linguistiques utilisées par le modèle COCOMO'81.

Nous présentons, dans le quatrième chapitre, les trois étapes principales constituant le processus de formulation d'une estimation des coûts de *Fuzzy Analogy*: Identification des projets logiciels, Évaluation de la similarité entre les projets logiciels et Adaptation. Pour chaque étape, nous présentons les différentes solutions que nous adoptons pour la tolérance des imprécisions. Spécifiquement pour l'étape d'Adaptation, nous présentons la stratégie de gestion des incertitudes au niveau des estimations fournies par *Fuzzy Analogy*. Le chapitre présente aussi les résultats d'une comparaison de la performance de *Fuzzy Analogy* avec celles de trois autres techniques d'estimation des coûts en utilisant les projets du COCOMO'81.

Le cinquième chapitre présente l'intégration de la caractéristique d'apprentissage dans *Fuzzy Analogy*. Ainsi, nous présentons, dans un premier temps, les différentes fonctionnalités de notre prototype logiciel F_ANGEL relatives à la mise à jour des différents paramètres de *Fuzzy Analogy*. Ensuite, nous étudions l'utilisation des réseaux de neurones pour incorporer un apprentissage automatique dans F_ANGEL. Nous expérimentons en particulier le cas d'un réseau RBFN.

La conclusion générale de ce document dresse un bilan des recherches entreprises dans cette thèse, nos contributions principales dans différents domaines de recherche et enfin, nos perspectives dans le domaine des applications de l'intelligence artificielle en génie logiciel.

# CHAPITRE I

# LES TECHNIQUES D'ESTIMATION DES COÛTS DE DÉVELOPPEMENT DE LOGICIELS

Dans ce chapitre, nous proposons une revue de la littérature sur les différentes techniques d'estimation des coûts de développement de logiciels. Ainsi, nous présentons brièvement celles se basant sur l'expertise, l'analogie et la stratégie descendante ou ascendante. Dans ce chapitre, l'emphase est mise sur la technique d'estimation se basant sur la modélisation. Nous exposons, en particulier, deux approches pour la modélisation de la relation exprimant l'effort de développement en fonction des facteurs affectant le coût: l'approche *paramétrique* utilisant des techniques statistiques telles que la régression linéaire et l'analyse bayesienne et l'approche *non-paramétrique* utilisant des techniques de l'Intelligence Artificielle telles que les réseaux de neurones, les arbres de décision et les algorithmes génétiques. Pour chaque technique de modélisation, nous présentons les avantages et les inconvénients. Nous terminons ce chapitre par une synthèse sur les travaux de recherche ayant comme objectif la validation de ces techniques sur des bases de projets logiciels historiques.

## 1.1 INTRODUCTION

Le contrôle et la maîtrise du processus de développement d'un logiciel sont basés sur une estimation fiable de son coût. En effet, sans cette estimation, les gestionnaires et les autres responsables d'un projet logiciel ne pourront assigner les budgets et les compétences nécessaires afin de mener à bien le développement du projet. Ainsi, les gestionnaires, s'appuyant généralement sur les résultats des estimations, pourraient commettre des erreurs irréparables, en prévoyant des plans inadéquats ou en assignant au projet des budgets insuffisants. Ces erreurs pourraient aussi se répercuter sur les utilisateurs. En effet, un allongement du délai de développement initialement prévu du logiciel entraînerait un retard

de livraison et d'installation du logiciel. Les utilisateurs seraient donc confrontés à des problèmes d'aspects économique et organisationnel, surtout si, comme c'est le cas en général, ils avaient dressé leurs plannings et mis en place une nouvelle structure en affectant du personnel et/ou en aménageant les locaux par exemple.

Bien que des chiffres précis soient difficiles à établir, il est de plus en plus admis que les coûts de développement de logiciels représentent une part importante des coûts informatiques. Ainsi, dans beaucoup de projets logiciels, le coût de développement dépasse 80% du coût total initial du projet (Boehm, 1981). La figure 1.1 résume la situation de l'évolution des coûts de développement par rapport à ceux d'achat de matériel dans un projet logiciel (Boehm, 1981). La croissance de ces coûts ainsi que les difficultés que l'on rencontre pour les prévoir et les contrôler constituent encore aujourd'hui une préoccupation des chercheurs et des responsables informatiques. En effet, ces responsables doivent faire face aux besoins sans cesse grandissants et diversifiés des utilisateurs mais aussi, aux différents problèmes rencontrés au cours du développement d'un logiciel (retard dans les délais, coût croissant, qualité faible, etc.).

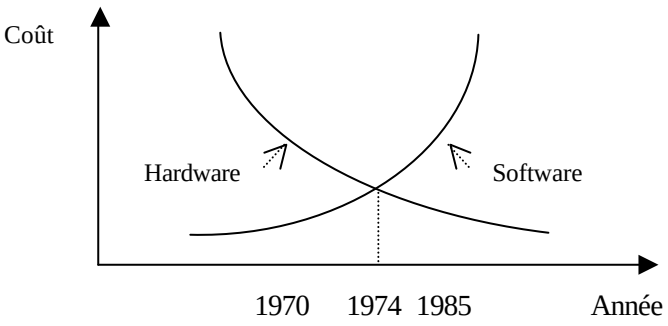



**Figure 1.1** Tendance de l'évolution des coûts de développement vs les coûts d'achat de matériel dans un projet logiciel (Boehm, 1981).

Vu l'importance d'une estimation fiable du coût dans la réussite d'un projet logiciel, le domaine d'estimation des coûts de logiciels a été l'objet de plusieurs travaux de recherche ainsi que d'une étroite collaboration entre les chercheurs et les industriels afin de développer et de valider des approches d'estimation propres au domaine du génie logiciel. D'ailleurs, nous pouvons constater que le domaine d'estimation des coûts, à travers sa littérature, est

parmi les plus fertiles en génie logiciel. L'estimation du coût d'un projet logiciel comporte l'étude des trois attributs suivants: l'*effort*, le *temps* et le *coût*. Ce sont ces trois attributs qu'on désigne souvent dans la littérature par les aspects économiques d'un projet logiciel.

## 1.2 LA MESURE HOMME-MOIS

L'effort de développement d'un logiciel représente le nombre de personnes ainsi que le nombre de mois nécessaires pour achever le projet. Par exemple, si 10 personnes ont travaillé pendant 5 mois dans un projet, l'effort de développement de ce projet est alors 50 Homme-mois. Par conséquent, si on connaît les salaires mensuels de ces intervenants dans le projet, son coût est tout simplement la somme des salaires multipliés par le nombre de mois de travail. Ceci implique donc que les trois attributs: *effort*, *temps* et *coût*, sont étroitement liés. En effet, la majeure partie des coûts de développement d'un logiciel correspond aux frais salariaux des ingénieurs logiciels. Ainsi, le coût d'un projet logiciel est directement proportionnel au nombre d'Homme-mois nécessaires à la réalisation du projet. Il y a, évidemment, d'autres coûts, par exemple le matériel, les déplacements et les formations du personnel impliqué; mais ceux-ci sont moins difficiles à établir. En effet, ils ne sont pas basés sur des impondérables tels que la productivité du personnel impliqué dans le projet ou la taille du logiciel à développer.

Du fait que le coût de développement d'un logiciel est déterminé par le nombre de personnes et de mois de travail, c'est-à-dire l'effort de développement, les deux terminologies d'estimation des coûts et d'*estimation de l'effort* sont considérées équivalentes en génie logiciel. En général, la plupart des approches d'estimation des coûts de logiciels, déterminent, en première étape, l'effort de développement; ensuite, elles évaluent le coût en utilisant les montants salariaux du personnel impliqué. L'unité *Homme-mois*, utilisée souvent pour mesurer l'effort de développement d'un logiciel, a fait l'objet d'une critique pertinente de Brooks dans son article: *The Mythical Man-Months* (Brooks, 1975):

> «The man-months as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable. Men and months are interchangeable commodities only when a task can be partionned among many workers with no communication among them»

Brooks a mené son étude sur un des plus gros logiciels de l'époque, OS/360, et a déduit plusieurs résultats qui ont pu être confirmés par plusieurs expérimentations. L'un de ses résultats principaux a été que les hommes et les mois ne sont pas interchangeables. Ceci est venu réfuter l'hypothèse courante qui affirmait jadis qu'en réduisant le délai normal de développement d'un logiciel d'un facteur donné, les ressources humaines nécessaires étaient tout simplement multipliées par ce facteur. Cette hypothèse reste valable dans certaines industries classiques où une tâche, relevant du processus de développement, peut être facilement divisée et réalisée par un ensemble de personnes sans qu'il y ait de la communication entre ces personnes. Ainsi, dans de tels cas, l'ajout du personnel dans le projet permet de réduire son temps de développement. Cependant, en industrie de logiciels, ce cas est rarement rencontré. En effet, les activités de développement d'un logiciel ne sont souvent pas partitionnables du fait de leur caractère séquentiel. Elles sont parfois partitionnables avec un taux de communications très élevé entre les intervenants, et très rarement sans communications intenses entre les intervenants (Fig. 1.2).

## 1.3 TECHNIQUES D'ESTIMATION DES COÛTS

Un inventaire des diverses techniques d'estimation des coûts est donné par Boehm (Boehm, 1981). Il identifie sept techniques différentes. Celles-ci sont basées sur l'expertise, l'estimation par analogie, la loi de Parkinson (un projet coûtera ce que vous avez décidé de dépenser), le prix du client (un projet coûtera ce que le client est disposé à dépenser), l'estimation descendante, l'estimation ascendante et la modélisation. Cependant, selon Fenton et Pfleeger, la loi de Parkinson et le Prix du client ne peuvent être considérés en tant que techniques d'estimation des coûts; Fenton et Pfleeger considèrent que le prix du client est une contrainte au projet qui peut déterminer sa faisabilité tandis que la loi de Parkinson exprime un but plutôt qu'une estimation (Fenton et Pfleeger, 1997). Chacune des techniques citées ci-dessus a des avantages et des inconvénients, mais le plus important selon Boehm, est qu'aucune technique ne peut être appliquée de façon exclusive. Dans ce qui suit, nous présentons brièvement les techniques d'estimation basées sur le jugement de l'expert, l'analogie, l'estimation ascendante et l'estimation descendante. Ensuite, l'accent sera mis sur la technique d'estimation basée sur la modélisation (Sect. 1.4).

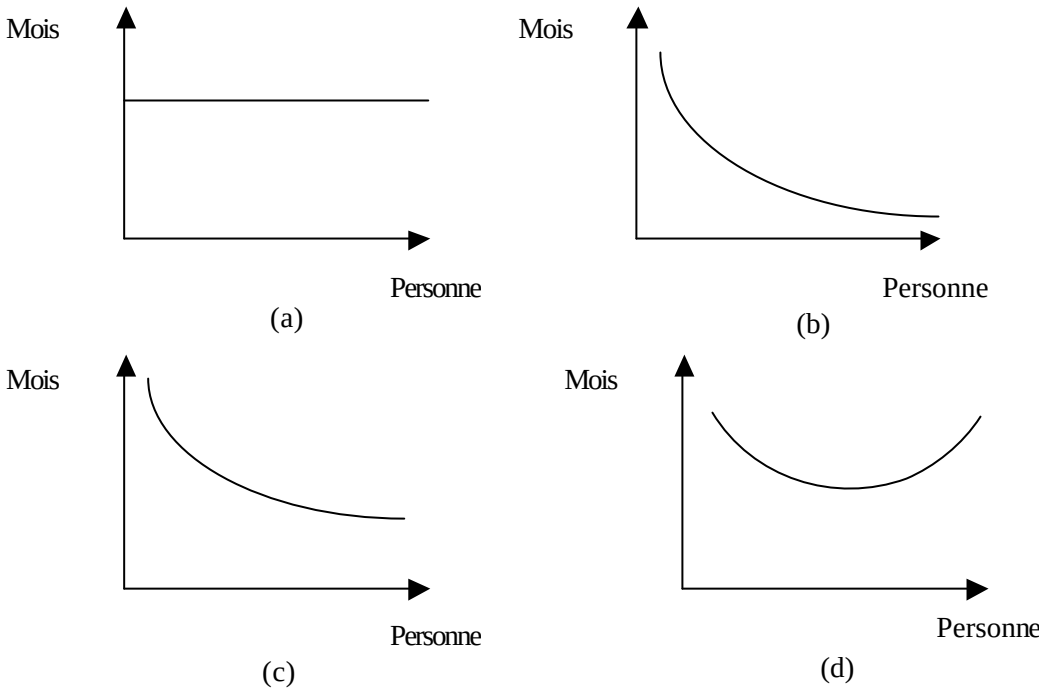



**Figure 1.2** Effets de la répartition d'une tâche sur plusieurs personnes. (a) Tâche non partitionable: peu importe le nombre de personnes, le temps de développement reste le même. (b) Tâche facilement partitionnable: le nombre de personnes réduit énormément le temps de développement. (c) Tâche partitionnable avec un taux faible de communications entre les intervenants: le nombre de personnes réduit moyennement le temps de développement. (d) Tâche partitionnable avec un taux de communications très élevé entre les intervenants: le nombre de personnes allonge le temps de développement du fait que l'effort de communications ajouté peut contrer entièrement la répartition de la tâche d'origine (Brooks, 1975).

### 1.3.1 Jugement de l'expert

Cette technique consiste à consulter un ou plusieurs experts qui utilisent leurs expériences ainsi que leur compréhension du projet afin de fournir une estimation à son coût. Elle était parmi les premières techniques utilisées en estimation des coûts. Il est préférable d'avoir plusieurs valeurs estimées du coût émanant de plusieurs experts pour alléger les problèmes de subjectivité, de pessimisme et d'optimisme (Boehm, 1981). Il y a plusieurs façons de combiner les différentes valeurs estimées du coût d'un projet logiciel. La plus simple est

d'utiliser une des méthodes statistiques telles que la moyenne, la médiane ou le mode de toutes les valeurs pour déterminer une estimation du coût. Cette approche présente l'inconvénient d'être sujette aux préjugés des estimations extrêmes. Une autre méthode, référée souvent dans la littérature par la méthode Delphi, consiste à réitérer ce processus d'estimation plusieurs fois jusqu'à ce qu'un consensus soit établi. L'idée principale de la méthode Delphi est qu'un coordinateur distribue une description du projet logiciel (souvent le dossier d'analyse des besoins du logiciel) ainsi qu'un formulaire d'estimation aux experts; ensuite, les experts remplissent les formulaires et les renvoient au coordinateur. Ce dernier pourra, dépendamment de la version de la méthode Delphi, convoquer une réunion des experts pour discuter des différentes estimations. Ce processus est réitéré jusqu'à l'adoption d'une estimation par tout le groupe.

### 1.3.2 Estimation par analogie

Dans cette technique d'estimation, l'évaluateur compare le projet logiciel dont on veut estimer le coût avec des projets logiciels déjà achevés. Il doit identifier les similarités et les différences entre le nouveau projet et tous les anciens projets. Pour cela, il doit auparavant décrire les projets logiciels par un ensemble de caractéristiques. Les ressemblances entre les caractéristiques du nouveau projet avec celles des projets achevés déterminent les degrés de similarité entre les projets. Ensuite, il utilise ces degrés de similarité pour déterminer une estimation au coût du nouveau projet.

Dans sa première version informelle, l'estimation par analogie a été considérée comme une alternative de l'estimation par les jugements des experts. En effet, les experts raisonnent souvent par analogie quand on leur demande une évaluation du coût d'un logiciel à partir de son dossier d'analyse des besoins. Ainsi, les experts utilisent leurs connaissances sur des projets déjà achevés et déterminent implicitement les similarités et les différences entre le nouveau projet et les projets achevés. Cependant, plusieurs versions formelles de l'estimation par analogie ont été récemment développées (Vicinanza et Prietula, 1990; Malabocchia, 1995; Shepperd et Schofield, 1997). Par ce fait, on la considère de plus en plus comme une technique d'estimation qui se base sur la modélisation. D'ailleurs, le modèle d'estimation des coûts, que nous développons dans cette thèse, adopte une version floue de l'estimation par

analogie. Le quatrième chapitre décrira l'état de l'art des versions formelles de l'estimation par analogie ainsi que notre nouvelle modélisation de l'estimation par analogie. Notre modèle d'estimation par analogie aura comme avantage la prise en considération de la tolérance des imprécisions, la gestion des incertitudes et l'apprentissage.

### 1.3.3 Estimation ascendante et descendante

Dans l'estimation ascendante, le projet logiciel (ou le logiciel) est décomposé en plusieurs tâches (ou composantes) constituant ainsi une arborescence de toutes les tâches (ou composantes) du projet logiciel (ou du logiciel). Tout d'abord, on estime l'effort nécessaire pour chaque tâche du plus bas niveau dans l'arborescence; ensuite, on détermine progressivement l'effort des autres tâches, se retrouvant dans un niveau supérieur dans l'arborescence, en combinant les efforts nécessaires associés aux sous-tâches. En général, l'effort nécessaire d'une tâche sera la somme de ceux de ses sous-tâches. Cependant, dans le cas des projets logiciels complexes, d'autres techniques plus sophistiquées, telles que celles se basant sur des formules mathématiques typiques ou sur des règles d'induction (si-alors), peuvent être utilisées afin de mettre en valeur la complexité des interfaces de communication entre les tâches.

Dans l'estimation descendante, on évalue une estimation globale de l'effort de tout le projet logiciel; ensuite, on répartit cet effort total sur toutes les tâches du projet logiciel. Dans les deux cas de l'estimation ascendante ou descendante, les valeurs estimées de l'effort sont déterminées en utilisant la technique du jugement de l'expert, l'estimation par analogie ou un modèle d'estimation.

## 1.4 LES MODÈLES D'ESTIMATION DES COÛTS

Les techniques d'estimation, basées sur la modélisation, ont été développées pour pallier aux inconvénients des techniques informelles se basant essentiellement sur la disponibilité des experts. En effet, la non-disponibilité des experts cause, dans la plupart des situations, des retards dans le processus d'estimation des coûts. En plus, le processus d'estimation des coûts d'une technique informelle est très ardu du fait qu'il implique le facteur humain tout au long

de la prise de décision. La modélisation s'avère donc une approche prometteuse pour la mise au point d'une technique formelle d'estimation des coûts. Elle a les avantages suivants sur les autres techniques d'estimation:

- Elle permet de rationaliser le processus d'estimation. En effet, la modélisation fournit un processus d'estimation clair, autrement dit, un processus opérationnel (praticable).

- Elle offre plusieurs possibilités aux évaluateurs afin d'améliorer la précision des estimations fournies. En effet, les évaluateurs pourront ajouter, supprimer et/ou modifier les paramètres du modèle et analyser ainsi les impacts de ces changements sur la précision des estimations.

- Elle offre une aide précieuse pour une bonne gestion d'un projet logiciel (formules de répartition de l'effort total sur toutes les phases de développement, mécanismes de prise de décision en cas d'insuffisance du budget alloué au développement, formules du comptage du code réutilisé, etc.).

- Elle permet l'automatisation (la programmation) du processus d'estimation (logiciels d'estimation des coûts).

Essentiellement, les modèles d'estimation des coûts se basent d'abord sur l'identification des facteurs influençant les coûts de développement de logiciels (*cost drivers*) et deuxièmement sur l'identification de la nature de la relation exprimant l'effort en fonction de ces facteurs. L'attribut *taille du logiciel* a été souvent considéré comme étant le facteur le plus significatif pour l'estimation de l'effort de développement. Cependant, d'autres facteurs tel que l'expérience du personnel (analystes, concepteurs, programmeurs, etc.) impliqué dans le projet, les méthodes utilisées dans le développement, et le degré de réutilisation ont été aussi pris en compte dans la plupart des modèles d'estimation quand la taille d'un logiciel était insuffisante pour expliquer les variations des efforts de développement des projets logiciels.

L'identification de la nature de la relation exprimant l'effort de développement en fonction des facteurs du coût représente l'étape principale dans le processus de mise au point d'un modèle d'estimation des coûts. Cette relation sera représentée par la fonction $f$:

$$Effort = f(X_1, X_2, ..., X_M) \qquad \text{(Équation 1.1)}$$

où $X_1$, $X_2$ et $X_M$ sont les facteurs affectant le coût de développement; $f$ est la relation exprimant l'effort en fonction des facteurs $X_i$.

La détermination de la fonction $f$ est souvent basée sur l'analyse des données historiques collectées sur les projets logiciels déjà achevés. Ces données historiques peuvent être représentées par une matrice N× (M+1):

$$\begin{pmatrix} & \textbf{Effort réel} & X_1 & X_2 & \dots & X_M \\ \textbf{Projet 1} & E_1 & x_{11} & x_{12} & \dots & x_{1M} \\ \textbf{Projet 2} & E_2 & x_{21} & x_{22} & \dots & x_{2M} \\ \textbf{Projet 3} & E_3 & x_{31} & x_{32} & \dots & x_{3M} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \textbf{Projet N} & E_N & x_{N1} & x_{N2} & \dots & x_{NM} \end{pmatrix} \qquad \text{(Matrice 1.1)}$$

où N représente le nombre de projets logiciels déjà achevés et M le nombre de variables décrivant les projets logiciels (facteurs affectant le coût). La colonne *Effort réel* représente l'effort réel de chaque projet logiciel déjà achevé, $E_i$. Les données historiques sur des projets logiciels déjà achevés représentent des exemples de la relation exprimant l'effort en fonction des facteurs du coût. Le défi est donc de reconstruire cette relation à partir de ces exemples afin qu'elle soit utilisée pour prédire le coût des nouveaux projets logiciels (nouvelles données).

Il existe plusieurs techniques qui permettent la reconstruction de la fonction $f$ à partir d'un ensemble d'exemples (associations entrées - sorties). Ces techniques proviennent principalement de trois domaines: les Statistiques, l'Approximation des fonctions et l'Intelligence artificielle. Néanmoins, nous pouvons les regrouper en deux catégories:

1- Les techniques *paramétriques*: elles présument au préalable que la fonction $f$ a une forme bien déterminée (linéaire ou analytique). Ensuite, elles déterminent les paramètres de la fonction $f$. Le nombre de paramètres de la fonction $f$ est connu d'avance. Des exemples de ces techniques sont la régression linéaire simple ou multiple, l'analyse bayesienne et l'interpolation polynomiale.

2- Les techniques *non-paramétriques*: elles ne fixent aucune forme à la fonction $f$. Elles développent des procédures d'approximation de la fonction $f$ sans connaître sa forme. Des exemples de ces techniques sont les réseaux de neurones, le raisonnement à base de cas (*Case-Based Reasoning* -CBR-), les arbres de régression et les règles d'induction (si-alors).

Nous décrivons, dans les deux sections suivantes, des exemples de modèles d'estimation des coûts mis au point en utilisant des techniques paramétriques et/ou non-paramétriques.

## 1.5 Les modèles paramétriques d'estimation des coûts

Les modèles paramétriques d'estimation des coûts de développement de logiciels se basent essentiellement sur une équation centrale exprimant l'effort en fonction d'un certain nombre de variables considérées comme étant les principales conductrices du coût. En plus, le modèle pourra utiliser des tables ou des formules prédéterminées pour évaluer les entrées de son équation centrale. La mise au point d'un modèle paramétrique d'estimation suppose donc que la forme de l'équation centrale du modèle est connue. Cette équation centrale représente la relation exprimant l'effort en fonction des facteurs affectant les coûts.

Du fait que les techniques paramétriques étaient les premières appliquées dans le domaine de l'estimation des coûts, plusieurs modèles paramétriques ont été développés. Des exemples de ces modèles sont le modèle d'Halstead (Halstead, 1975), IBM-FSD (Walston et Felix, 1977) PUTNAM-SLIM (Putnam, 1978), COCOMO'81 (Boehm, 1981), COCOMOII (Boehm, 1995), et ceux se basant sur les points de fonctions (Albrecht et Gaffney, 1983; Abran et Robillard, 1994; Matson, Barrett et Mellichamp, 1994). Les modèles paramétriques peuvent être classifiés selon trois critères:

a) La forme de l'équation centrale du modèle (linéaire ou analytique);

b) le modèle est-il Statique ou Dynamique? Un modèle est dit Dynamique si son équation centrale contient la variable *temps*. Ainsi le modèle fournit, en plus de l'effort total, l'effort instantané durant tout le processus de développement. Un modèle est dit Statique si son équation centrale ne contient pas la variable *temps*;

c) le modèle est-il Global ou Détaillé? Un modèle est dit Détaillé si, en plus de l'effort total, il fournit aussi sa répartition sur toutes les activités et les phases du cycle de développement. Notamment, on peut remarquer qu'un modèle Dynamique est nécessairement Détaillé. Un modèle Global ne fournit que l'effort total. Un modèle Statique n'est pas nécessairement un modèle Global.

### 1.5.1 Les modèles linéaires

Ces modèles adoptent une équation centrale linéaire de la forme:

$$Effort = a_0 + a_1 x_1 + a_2 x_2 + ... + a_M x_M \qquad \text{(Équation 1.2)}$$

où les $x_i$ sont les facteurs affectant l'effort et les $a_i$ sont des coefficients choisis pour fournir le meilleur ajustement à l'ensemble des données historiques observées. La mise au point de tels modèles est souvent accomplie en utilisant les techniques statistiques de régression linéaire simple ou multiple selon le nombre de variables $x_i$. La régression linéaire consiste à analyser un ensemble de données sur des projets logiciels déjà achevés (comme celles de la matrice 1.1) afin de reconstruire la relation, supposée être linéaire, exprimant l'effort (variable dépendante) en fonction des variables $x_i$ (variables indépendantes). Dans ce qui suit, nous présentons l'état de l'art sur les applications de la régression linéaire simple et multiple pour la mise au point des modèles linéaires d'estimation des coûts.

Régression simple

L'application de la régression linéaire simple, en estimation des coûts, a souvent considéré la taille du logiciel comme étant la variable indépendante la plus significative pour la prédiction de l'effort:

$$Effort = A + B \times taille \qquad \text{(Équation 1.3)}$$

où $A$ et $B$ sont des constantes. La taille d'un logiciel est souvent mesurée par le nombre de lignes de son code source (*Kilo Line Of Code* -KLOC-) ou le nombre de fonctionnalités qu'il offre à ses utilisateurs, calculé par la méthode des points de fonctions (*Function Points* -FP-; Albrecht et Gaffney, 1983; Abran et Robillard, 1994). Les deux paramètres $A$ et $B$ sont généralement déterminés en utilisant la méthode des moindres carrés qui consiste à chercher le minimum de la somme des carrés des résidus $e_i$:

$$F(a,b) = \sum_{i=1}^{n} e_i^{\,2} = \sum_{i=1}^{n} (Effort_{i,réel} - A - B \times taille_i)^2 \qquad \text{(Équation 1.4)}$$

Le résidu $e_i$ représente, pour chaque projet $P_i$, la différence entre la valeur réelle de l'effort de développement de $P_i$, $Effort_{i,réel}$, et la valeur estimée de son effort par le modèle linéaire de l'équation 1.3. Les valeurs $A$ et $B$ qui minimisent la fonction $F(a,b)$ sont celles adoptées par le modèle linéaire de l'équation 1.3.

Après la détermination des deux paramètres du modèle, $A$ et $B$, à partir de la base de données des projets logiciels, on teste la qualité de l'ajustement obtenu par le modèle. L'indicateur le plus couramment employé est le coefficient de détermination, $R^2$ représentant la proportion de la variabilité de l'effort expliquée par la taille; $R^2$ est compris entre 0 et 1; une valeur de $R^2$ proche de 1 implique que la taille explique bien l'effort d'un projet logiciel. Cependant, d'autres indicateurs relevant de la distribution des résidus $e_i$, et surtout des résidus standardisés $e_{ir}$, sont nécessaires pour détecter les défaillances d'un modèle linéaire ayant un $R^2$ voisin de 1. En effet, les résidus standardisés doivent être distribués indépendamment selon une loi gaussienne centrée réduite (moyenne=0 et variance=1). Ces suppositions sur les résidus standardisés d'un modèle linéaire sont examinées par au moins deux graphiques

(Saporta, 1990): 1) Le graphique des $e_{ir}$ en fonction de la taille ou l'effort estimé pour vérifier l'indépendance de la distribution des $e_{ir}$; 2) Le graphique des $e_{ir}$ pour vérifier la normalité de la distribution des $e_{ir}$. Ainsi, si l'une des deux suppositions n'est pas satisfaite, une transformation de la variable indépendante (taille) et/ou de la variable dépendante (effort) est nécessaire pour rendre linéaire la relation exprimant l'effort en fonction de la taille. Autrement, la relation serait considérée comme étant non-linéaire et par conséquent la méthode de la régression linéaire est inadaptée pour la mise au point du modèle.

La transformation logarithmique est la plus utilisée en estimation des coûts du fait que les modèles adoptent souvent une équation centrale de la forme :

$$Effort = B \times taille^{C} \qquad \text{(Équation 1.5)}$$

Pour rendre linéaire ce genre de modèles, on transforme les deux variables *effort* et *taille* en *log(effort)* et *log(taille)*. Ainsi, l'équation centrale transformée du modèle s'écrit comme suit:

$$log(Effort) = log(B) + C \times log(taille) \qquad \text{(Équation 1.6)}$$

Par conséquent, on utilise la régression linéaire pour déterminer les deux constantes *log(B)* et C qui minimisent la somme:

$$\sum ( log(effort_{réel} - log(B) - C \times log(taille))^{2} \qquad \text{(Équation 1.7)}$$

On détermine ainsi les deux constantes initiales $B$ et $C$ de l'équation centrale originale du modèle (Équation 1.5). Il faut noter que la minimisation de l'équation 1.7 n'entraîne pas nécessairement la minimisation de la somme des différences entre les efforts réels et estimés exprimés sur l'échelle originale. Cependant, des résultats satisfaisants sont souvent obtenus (Box et Cox, 1964; Draper et Smith, 1981; Morrison, 1983). Des exemples de modèles, mis au point en appliquant la régression linéaire après la transformation logarithmique des deux variables *effort* et la *taille*, sont (Conte, Dunsmore et Shen, 1986):

$$Effort = 5.2 \times KLOC^{0.91} \qquad Walston-Felix$$
$$Effort = 3.2 \times KLOC^{1.05} \qquad COCOMO\ simple$$
$$Effort = 5.288 \times KLOC^{1.047} \qquad Doty\ (for\ KLOC > 9)$$

En plus de la transformation logarithmique, d'autres transformations ont aussi été appliquées pour rendre linéaire la relation exprimant l'effort en fonction de la taille, telle que celle utilisée par Matson, Barrett et Mellichamp dans leurs expérimentations sur la base de données d'Albrecht et Gaffney (Matson, Barrett et Mellichamp, 1994). Cette base de données contient 24 projets logiciels (Albrecht et Gaffney, 1983). Pour chaque projet, ils fournissent le nombre FP associé ainsi que son effort de développement en Homme-heures. Le nombre FP est calculé en fonction de cinq paramètres: les entrées (IN), les sorties (OUT), les questions (INQ), les fichiers (FILE) et le facteur d'ajustement (ADJ). L'ajustement obtenu par la régression simple sur les données non transformées (*effort*, *FP*) indique un coefficient de détermination élevé, $R^2$=87,4% (Fig. 1.3a). Cependant, le graphe des résidus standardisés en fonction de l'effort estimé montre une dépendance évidente entre le nombre FP et le résidu standardisé d'un projet. En effet, le résidu standardisé semble être croissant en fonction du nombre FP: quatre des cinq projets ayant des nombres FP élevés, ont aussi des résidus standardisés élevés (cas {1}, {2}, {19}, {20} de la figure 1.3b). Ceci contredit une supposition fondamentale de la régression simple qui postule que la distribution des résidus standardisés est indépendante des deux variables *effort* et *FP*. Pour remédier à cette anomalie, Matson, Barrett et Mellichamp ont adopté la transformation *effort*→*Racine(effort)*. Ainsi, ils ont appliqué la régression simple aux données transformées (*Racine(effort)*, *FP*). L'ajustement obtenu indique un coefficient de détermination élevé, $R^2$= 89,9%, et une indépendance de la distribution des résidus standardisés des deux variables *Racine(effort)* et *FP* (Fig. 1.3c et Fig. 1.3d).

Régression linéaire multiple

La régression linéaire simple est utilisée pour la mise au point d'un modèle linéaire d'estimation dans le cas où le nombre de facteurs considérés dans le modèle est égal à un (souvent la taille du logiciel). Dans le cas où le nombre de ces facteurs est supérieur ou égal à deux, la mise au point du modèle fait appel à la régression linéaire multiple. Des exemples de

facteurs affectant le coût, autres que la taille, sont l'expérience du personnel impliqué dans le développement, la complexité de l'application et la méthodologie de développement. En général, ces facteurs jouent un rôle d'ajustement de l'équation, dite Nominale, exprimant l'effort seulement en fonction de la taille du logiciel. En effet, Boehm avait constaté que des logiciels ayant les mêmes tailles ne requéraient pas nécessairement les mêmes efforts de développement (Boehm, 1981). La prise en compte d'autres facteurs dans l'équation centrale d'un modèle linéaire d'estimation a donc pour objectif l'explication de la variation des coûts de deux projets ayant les mêmes tailles.

L'application de la régression linéaire multiple en estimation des coûts permet la mise au point de modèles linéaires ayant une équation centrale comme celle de l'équation 1.2. Les paramètres $a_i$ de l'équation sont ceux qui minimisent la somme des carrés:

$$\sum_{i=1}^{n} e_i^{\,2} = \sum_{i=1}^{n} (Effort_{i,réel} - a_0 - a_1 x_1^i - a_2 x_2^i - .. - a_M x_M^i)^2 \qquad \text{(Équation 1.8)}$$

Les critères utilisés dans l'évaluation de la qualité de l'ajustement, dans le cas de la régression linéaire simple, sont aussi applicables dans le cas de la régression linéaire multiple. En effet, un coefficient de détermination $R^2$ proche de un signifie que l'ajustement est convenable; aussi, l'analyse des résidus $e_{ir}$ permet de vérifier les suppositions du modèle linéaire sur l'indépendance de la distribution des $e_{ir}$ dont les graphes en fonction de la variable dépendante (effort) et les variables indépendantes ($x_i$) du modèle ne doivent montrer aucune tendance. En plus, il y a deux autres critères d'évaluation propres au cas de la régression linéaire multiple (Saporta, 1990):

- Le premier consiste à mesurer l'impact de la multicolinéarité sur l'instabilité du modèle. En effet, si les facteurs influençant le coût de développement $x_i$ sont très corrélés entre eux, les estimations du modèle seront entachées d'erreurs considérables même si $R^2$ a une valeur proche de 1. En général, l'étude de l'effet de la colinéarité entre les facteurs $x_i$ s'effectue au moyen des facteurs d'inflation de la variance et des valeurs propres de la matrice de corrélation entre les $x_i$.

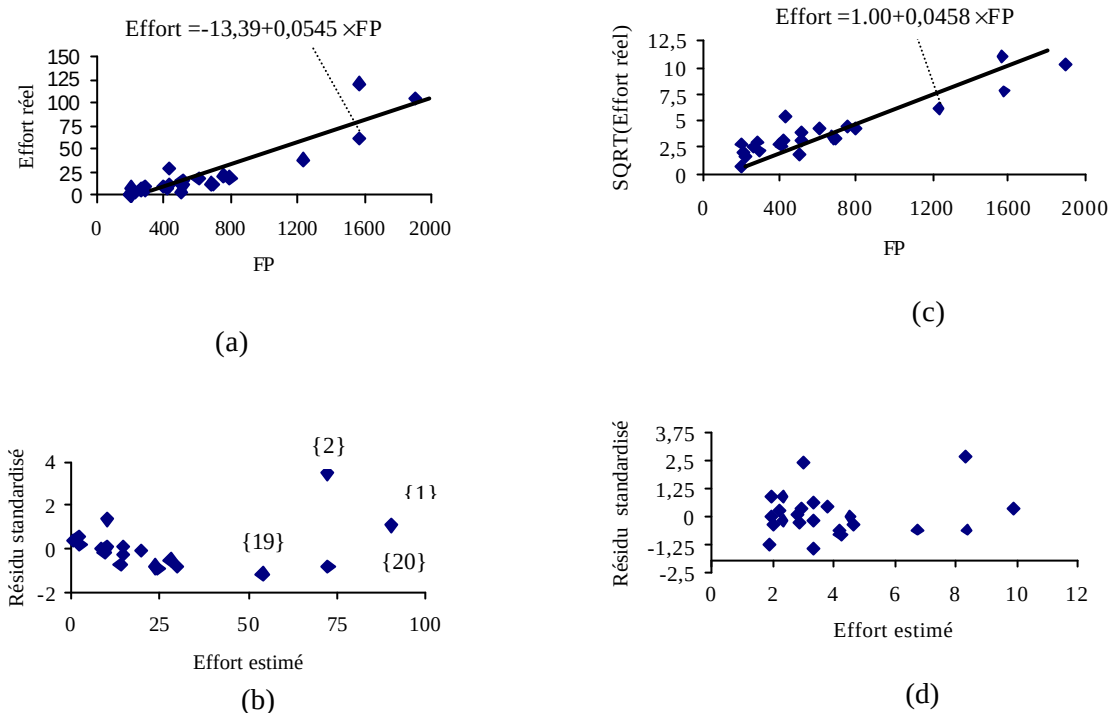



**Figure 1.3** Analyse de l'application de la régression linéaire simple à la base de projets logiciels d'Albrecht. (a) Effort réel vs taille mesurée en FP. (b) Résidu standardisé vs Effort estimé du modèle de (a). (c) Racine (effort réel) vs FP. (d) Résidu standardisé vs Effort estimé du modèle de (b) (Matson, Barrett et Mellichamp, 1994).

- Le deuxième consiste à déterminer le sous-ensemble de tous les facteurs $x_i$ qui donne des estimations satisfaisantes des coûts. Cela permettra au modèle d'éliminer les effets des facteurs non significatifs et redondants. On procède souvent par élimination successive ou par ajout successif des facteurs dans le modèle. Dans la première stratégie, on élimine celui qui provoque la diminution la plus faible de $R^2$; dans la deuxième stratégie, on ajoute celui qui provoque la progression la plus élevée de $R^2$. Dans les deux cas, un test d'arrêt est obligatoire. Une autre variante souvent utilisée en estimation des coûts est la méthode dite *Stepwise Regression.* Elle consiste à effectuer des tests de signification, à chaque étape, pour ne pas introduire une variable non significative et éliminer éventuellement des variables déjà introduites qui ne seraient plus informatives compte tenu de la dernière variable sélectionnée. La méthode s'arrête quand on ne peut ni ajouter ni retrancher des variables du modèle.

Comme dans le cas de la régression linéaire simple, souvent en estimation des coûts, la relation initiale exprimant l'effort en fonction des facteurs du coût n'est pas linéaire. La pratique consiste à utiliser la transformation logarithmique pour rendre linéaire cette relation initiale. Cette stratégie a souvent été adoptée par la communauté pour la calibration et pour l'adaptation des modèles linéaires d'estimation à des environnements différents de ceux à partir desquels ces modèles ont été initialement développés. Nous citons, en particulier, le cas du modèle COCOMO'81 de Boehm qui a été l'objet de plusieurs travaux de calibration et d'adaptation (Miyazaki et Mori, 1985; Marwane et Mili, 1991; Gulezian, 1991; Amrane et Slimani, 1993). Gulezian avait reformulé l'équation centrale initiale du modèle COCOMO'81 Intermédiaire afin d'utiliser la transformation logarithmique de toutes ses variables (dépendante et indépendantes) pour rendre le modèle linéaire (Gulezian, 1991):

$$Effort = a' {a'}_1^{m1} {a'}_3^{m3} (KISL)^{b'+b'_1 m_1 + b'_3 m_3} \prod_{i=1}^{16} {d'}_i^{C_{ij}-1} \qquad \text{(Équation 1.9)}$$

où KISL (Kilo Instructions Sources Livrées) mesure la taille du logiciel, les paramètres $a'_i$ et $b'_i$ représentent le mode du projet logiciel à développer, et les paramètres $d_i$ sont les coefficients associés aux facteurs affectant le coût autres que la taille du logiciel. La transformation logarithmique de l'équation 1.9 donne:

$$log(Effort) = log(a') + m_1 log(a'_1) + m_3 log(a'_3) + b' log(KISL) + b'_1 m_1 log(KISL)$$
$$+ b'_3 m_3 log(KISL) + \sum_{i=1}^{16} log(d'_i)(C_{ij} - 1)$$

Ainsi, Gulezian utilise la régression linéaire multiple pour déterminer les différents paramètres du modèle.

### 1.5.2 Les modèles analytiques

Dans la section précédente, nous avons étudié les modèles linéaires d'estimation, c'est-à-dire ceux qui adoptent une équation centrale linéaire, éventuellement après la transformation de la variable dépendante et/ou les variables indépendantes. Dans cette section, nous présentons une deuxième catégorie de modèles paramétrique: les modèles analytiques.

Les modèles analytiques adoptent une équation centrale non-linéaire. Leur utilisation est nécessaire quand la relation initiale, exprimant l'effort en fonction des facteurs du coût, n'est pas toujours linéaire et qu'aucune transformation des variables n'est possible afin de rendre cette relation linéaire (Fig. 1.4). Dans la littérature, on ne trouve que peu de modèles analytiques qui ont été développés (Halstead, 1975; Putnam, 1978). En effet:

- Souvent, un modèle linéaire satisfaisant et concurrent peut être mis en place, éventuellement après la transformation des variables.
- Quand cette relation n'est pas linéaire, l'identification de la forme de la relation exprimant l'effort en fonction des facteurs du coût est souvent très ardue.
- Les praticiens, dans leur majorité, préfèrent apporter des améliorations de détail au modèle linéaire concurrent plutôt que d'étudier le modèle non-linéaire initial.

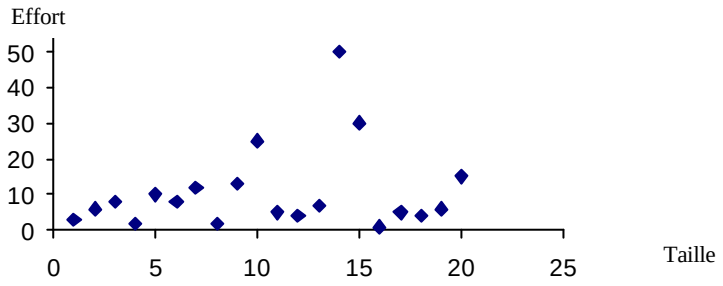



**Figure 1.4** Exemple où la relation Effort-taille n'est pas linéaire.

Dans ce qui suit, nous présentons brièvement deux exemples de modèles analytiques: le modèle d'Halstead et le modèle de Putnam.

Le modèle d'Halstead

En 1972, Maurice Halstead a entrepris une étude empirique des algorithmes afin de tester l'hypothèse selon laquelle, dans un programme, le comptage des opérateurs (expressions verbales) et des opérandes (noms ou désignations de données) pouvait faire apparaître des

corrélations avec le nombre des erreurs contenues dans les algorithmes (Halstead, 1975). Suite à cette étude empirique, Halstead avait défini quatre métriques basées sur l'analyse lexicale du code source:

$\mu_1$=nombre d'opérateurs distincts

$\mu_2$=nombre d'opérandes distincts

$N_1$=nombre total d'utilisation des opérateurs

$N_2$=nombre total d'utilisation des opérandes

Ensuite, Halstead avait utilisé ces quatre métriques pour développer une panoplie de formules d'évaluation de certains attributs d'un logiciel tels que le vocabulaire et le volume d'un programme, l'effort et la durée de codage d'un programme. Les deux équations en rapport avec l'effort et la durée de codage d'un programme sont respectivement:

$$Effort = \frac{m_1 N_2 N \log_2(m_1 + m_2)}{2m_2}$$
$$Durée = \frac{Effort}{18} \qquad \text{(Équation 1.10)}$$

où $N=N_1+N_2$. Cette équation d'estimation de l'effort à partir des quatre métriques de base d'Halstead n'a pas eu beaucoup d'intérêt dans la plupart des travaux de recherche en estimation des coûts du fait que plusieurs hypothèses adoptées par Halstead sont considérées comme erronées (Fenton et Pflegger, 1997). En plus, la plupart des mesures proposées par Halstead sont basées sur le code source d'un logiciel; les autres phases du cycle de développement telles que la spécification et la conception ne sont pas prises en considération bien qu'elles requièrent souvent un effort considérable.

Le modèle de Putnam

En 1978, Putnam a proposé un modèle d'estimation de l'effort de développement et de maintenance d'un logiciel en se basant sur les remarques et sur les conclusions que Norden avait déjà déduit en étudiant des projets de développement dans beaucoup de domaines autres

que le génie logiciel (Putnam, 1978). En effet, Norden avait observé que la distribution de l'effort total sur tout le cycle de développement d'un projet, dans plusieurs domaines, pouvait s'approcher avec une grande précision par la fonction de Rayleigh (Fig. 1.5). Ainsi, Putnam avait utilisé cette fonction pour exprimer la répartition de l'effort total de développement sur tout le cycle de vie d'un logiciel (développement et maintenance):

$$Effort(t) = CT(1 - e^{\frac{-1}{2}(\frac{t}{t_d})^2}) \qquad \text{(Équation 1.11)}$$

où *Effort(t)* représente l'effectif cumulé à l'instant t, *CT* est l'effort total de tout le cycle de vie du logiciel, et $t_d$ est le temps de développement.

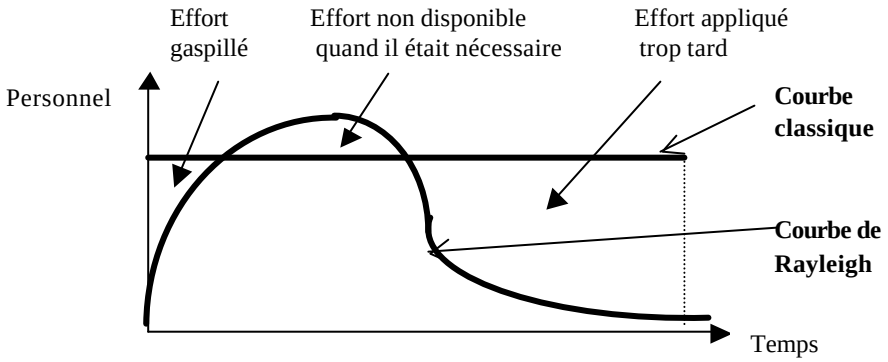



**Figure 1.5** Répartition de l'effort selon la courbe de Rayleigh et la courbe classique (Putnam, 1978).

Pour exprimer l'effort en fonction de la taille d'un logiciel, Putnam a constaté, après une étude empirique des projets réels, que la productivité dépend de la difficulté d'un projet selon l'équation suivante:

$$\frac{ISL}{Effort(t_d)} = kD^{-\frac{2}{3}} \qquad \text{(Équation 1.12)}$$

où *D* est la difficulté du projet, *ISL* le nombre d'instructions sources livrées, et *k* un coefficient. Ainsi, Putnam a proposé l'équation suivante pour l'effort total d'un projet logiciel:

$$ISL = E\, CT^{\frac{1}{3}}\, t_d^{\frac{4}{3}} \qquad \text{(Équation 1.13)}$$

où E est un facteur technologique représentant l'effet de 14 conducteurs de coût tels que l'effet du langage utilisé, la méthodologie de développement et la procédure d'assurance qualité utilisée. L'équation 1.12 indique que l'effort est inversement proportionnel au temps de développement à la puissance quatre. Ceci signifie qu'une réduction du temps de développement d'un projet, par exemple de 10%, impliquera une augmentation de 40% de l'effort total.

### 1.5.3 Limitations des modèles paramétriques

Les modèles paramétriques d'estimation des coûts (linéaires ou analytiques) supposent la connaissance de la forme de la relation exprimant l'effort en fonction des facteurs conducteurs du coût. Autrement dit, la forme de l'équation centrale du modèle est bien connue et il suffit donc de déterminer les paramètres de cette équation en utilisant des techniques statistiques ou des méthodes théoriques du domaine de l'approximation des fonctions. En estimation des coûts de logiciels, on détermine souvent la forme de l'équation centrale du modèle en utilisant des connaissances relevant de l'environnement étudié ou en adoptant des formes déjà établies dans d'autres environnements. Par exemple, dans le cas de la base de projets d'Albrecht, la relation entre l'effort et la taille mesurée en nombre FP semble être linéaire, éventuellement après la transformation de la variable *effort* (Fig. 1.3). Par conséquent, la forme adoptée de l'équation centrale du modèle candidat est linéaire. Dans le cas du modèle de Putnam, l'équation centrale a une forme non-linéaire du fait que Putnam avait adopté la courbe de Rayleigh pour exprimer la répartition de l'effort total sur tout le cycle de vie. Ainsi, la mise au point d'un modèle paramétrique d'estimation des coûts nécessite au préalable l'identification de la forme de son équation centrale. Cela représente une limitation majeure car souvent:

- La relation entre l'effort et les conducteurs du coût ne semble avoir, dans plusieurs cas, aucune forme prédéterminée (Fig. 1.4),

- les connaissances sur la relation exprimant l'effort en fonction des conducteurs du coût ne sont pas disponibles, et/ou

- Les connaissances disponibles sont imprécises et/ou incertaines.

Aussi, au fur et à mesure que le modèle linéaire est opérationnel dans un organisme, le modèle doit être capable d'apprendre afin de considérer les nouveaux changements survenus dans l'organisme au niveau du développement et/ou de la maintenance de logiciels. Malheureusement, les techniques statistiques classiques, comme la méthode de régression, n'offrent pas de mécanismes pour un apprentissage automatique. Cependant, elles adoptent des mécanismes élémentaires tels que la mise à jour de la base des données historiques, la mise à jour des contributions des observations, et la mise à jour des coefficients du modèle afin d'introduire l'apprentissage dans le modèle. Ces mécanismes sont insuffisants et très superficiels et ne peuvent doter le modèle d'un degré d'apprentissage acceptable.

Certaines limitations des modèles paramétriques sont spécifiques aux cas des modèles linéaires, en particulier à la technique de régression linéaire, comme la performance de généralisation, la présence des observations extrêmes et la contribution des observations dans la régression:

- La généralisation explique le comportement du modèle dans les cas de nouvelles observations. En estimation des coûts, elle représente donc la capacité du modèle à fournir des estimations acceptables pour de nouveaux projets logiciels. En général, il faut être prudent, quand on se sert d'un modèle linéaire, pour prédire le coût d'un projet logiciel qui s'éloigne des limites de l'échantillon des projets logiciels ayant servi à la mise au point du modèle linéaire. Bien qu'elle n'apparaisse pas dans les équations du modèle, cette précaution est obligatoire dans le cas des modèles linéaires. Par exemple, supposons qu'un modèle linéaire ne considère que la taille pour l'estimation de l'effort et que la taille de tous les logiciels de la base de données historiques est comprise entre 10KISL et 40KISL; dans ce cas, l'utilisation de ce modèle pour estimer le coût d'un logiciel de taille 100KISL pourra conduire à une estimation aberrante.

- La présence des observations extrêmes (*outliers*) avec des résidus assez élevés comme ceux des projets logiciels {1}, {2}, {19} et {20} de la figure 1.3b, influence la précision d'un modèle linéaire d'estimation. Dans la littérature, on ne les considère souvent pas dans la mise au point des modèles afin d'améliorer la précision du modèle linéaire. Cependant, à moins que l'observation ne représente des données erronées, il est très contesté de supprimer ce type d'observations et de ne pas les prendre en considération dans la mise au point du modèle (Dolado, 2001; Gray et MacDonell, 1997). Dans certains cas, on peut éviter le problème des observations extrêmes seulement par une transformation de la variable dépendante et/ou des variables indépendantes (Matson, Barrett et Mellichamp, 1994). Autrement, d'autres formes de modèles paramétriques (ou carrément non-paramétriques) doivent être considérées pour représenter la relation exprimant l'effort en fonction des conducteurs du coût.

- L'étude des contributions de projets logiciels historiques dans la stabilité du modèle linéaire est aussi un aspect très intéressant qui doit être pris en compte dans la mise au point du modèle linéaire. On évalue souvent la contribution de chaque projet logiciel en appliquant la technique de régression sans prendre en considération ce projet logiciel. Les projets logiciels, impliquant des variations importantes dans la détermination des coefficients du modèle, sont considérés comme influençant le plus la stabilité du modèle. Afin d'éviter les influences de ces observations, la pratique consiste à les éliminer ou à pondérer les observations dans la technique de régression. La deuxième solution semble être la meilleure bien qu'il n'y ait pas une technique standard pour l'affectation des poids aux observations.

Une autre limitation des modèles paramétriques d'estimation des coûts est qu'ils requièrent une adaptation ou une calibration quand on veut les appliquer dans des environnements différents de ceux où ils ont été développés. En effet, le modèle paramétrique représente l'état de l'environnement étudié à l'aide de la forme de son équation centrale ainsi que les conducteurs du coût qu'il considère. Cependant, vu que les environnements de développement et/ou de maintenance de logiciels n'ont pas généralement les mêmes

caractéristiques, l'application d'un modèle, initialement mis au point dans un environnement à un autre, n'est pas souhaitable. Des exemples de caractéristiques qui peuvent différencier deux environnements de développement de logiciels sont le type d'applications logicielles développées (Scientifique, Gestion, Business, etc.), les méthodes et les outils de développement utilisés, et les contraintes imposées sur l'application (fiabilité, portabilité, complexité, etc.).

Selon Gulizian, la calibration d'un modèle paramétrique est sa reformulation sur une base de données de projets logiciels d'un environnement autre que celui du modèle initial (Gulezian, 1991). Ainsi, les équations du modèle calibré, y compris son équation centrale, ont les mêmes structures que celles du modèle initial mais pour lesquelles les coefficients sont recalculés. L'adaptation d'un modèle paramétrique est la modification des formes de certaines de ses équations ainsi que l'ajout ou la suppression de facteurs affectant le coût dans le modèle. Souvent en estimation des coûts, les chercheurs et les industriels utilisaient ces deux techniques (calibration et adaptation) pour mettre au point des modèles propres à leurs environnements à partir de modèles déjà mis au point dans d'autres environnements. Bien que la calibration (ou l'adaptation) représente, dans certains cas, une solution au problème de l'utilisation d'un modèle paramétrique ailleurs que dans son environnement, elle reste insuffisante quand les différences entre les environnements sont très flagrantes.

## 1.6 LES MODÈLES NON-PARAMÉTRIQUES D'ESTIMATION DES COÛTS

Les modèles non-paramétriques d'estimation des coûts ont été développés pour remédier aux inconvénients cités ci-dessus dans les modèles paramétriques. Ils n'ont pas une équation centrale d'estimation de l'effort. Ainsi, les modèles non-paramétriques n'exigent au préalable aucune forme à la relation exprimant l'effort en fonction des conducteurs du coût. Ils modélisent cette relation en utilisant des techniques d'intelligence artificielle telles que le raisonnement à base de cas, les réseaux de neurones, les algorithmes génétiques, les systèmes à base de règles et les arbres de décision. Les modèles non-paramétriques représentent donc une alternative prometteuse quand la relation entre l'effort et les conducteurs du coût semble n'avoir aucune forme prédéfinie (Fig. 1.4). De ce fait, plusieurs travaux de recherche ont été menés pour développer des modèles d'estimation des coûts en se basant sur le raisonnement

à base de cas (Vicinanza et Prietulla, 1990; Bisio et Malabocchia, 1995; Shepperd, Schofield et Kitchenham, 1996; Shepperd et Schofield, 1997; Kadoda et al., 2000), les réseaux de neurones ( Samson, Ellison et Dugard, 1993; Serluca, 1995; Srinivasan et Fisher, 1995; Hughes, 1996; Wittig et Finnie, 1997; Shukla, 2000), les arbres de décision (Selby et Porter, 1988; Porter et Selby, 1990a; Porter et Selby, 1990b; Srinivasan et Fisher, 1995), et les algorithmes génétiques (Dolado, 2000; Burgess et Lefley, 2001). Dans cette section, nous présentons brièvement les modèles non-paramétriques utilisant les réseaux de neurones, les arbres de décision et les algorithmes génétiques. Cependant, ceux utilisant le raisonnement à base de cas (CBR) seront présentés dans le quatrième chapitre du fait que le modèle dit *intelligent*, que nous proposerons dans cette thèse, adopte une version floue du raisonnement à base de cas.

### 1.6.1 Modèles d'estimation utilisant les réseaux de neurones

La modélisation par les réseaux de neurones est inspirée de la structure biologique du cerveau humain. Un réseau de neurones est caractérisé par son architecture, son algorithme d'apprentissage et ses fonctions d'activation. Dans la littérature, plusieurs modèles de réseaux de neurones ont été développés (Hertz, Krogh et Palmer, 1991; Hayken, 1994; Negnevitsky, 2001; Perlovsky, 2001). Ils peuvent être classifiés en deux catégories principales: les réseaux de neurones *feedforward* où il n'y a aucune boucle dans l'architecture du réseau, et les réseaux *récurrents* où une ou plusieurs boucles récursives apparaissent dans l'architecture du réseau (Hertz, Krogh et Palmer, 1991). Le Perceptron multi-couches utilisant l'algorithme d'apprentissage de rétro-propagation est souvent le plus adopté en estimation des coûts de logiciels (Samson, Ellison et Dugard, 1993; Serluca, 1995; Srinivasan et Fisher, 1995; Jorgersen, 1995; Hughes, 1996; Wittig et Finnie, 1997; Shukla, 2000). Dans ce modèle, les neurones sont organisés en couches et chaque couche ne peut avoir des connexions que vers la couche suivante. La figure 1.6 illustre un exemple d'un tel réseau pour l'estimation de l'effort de développement de logiciels. Le réseau produit un résultat (effort) en propageant ses entrées initiales (facteurs du coût ou attributs du projet logiciel) à travers les différents neurones du réseau jusqu'à la sortie. Chaque neurone d'une couche calcule sa sortie en appliquant sa fonction d'activation à ses entrées. Souvent, la fonction d'activation d'un neurone est la fonction Sigmoïde définie par:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad \text{(Équation 1.14)}$$

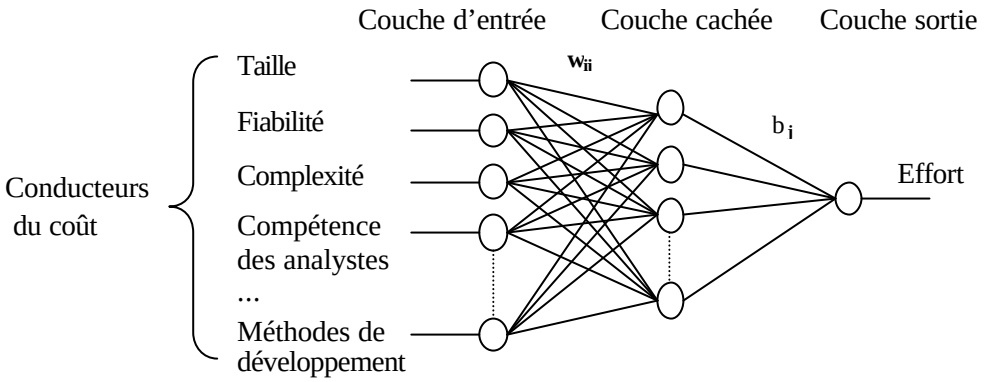



**Figure 1.6** Architecture d'un réseau de neurone à trois couches pour l'estimation de l'effort.

La configuration d'un réseau de neurones pour l'estimation de l'effort de développement de logiciels nécessite le choix d'une architecture adéquate, un algorithme d'apprentissage et des fonctions d'activation. Dans le cas d'un Perceptron multi-couches, la configuration se restreint au choix des différents paramètres de sa topologie tels que le nombre de couches cachées, le nombre de neurones par couche, et les valeurs initiales des poids de connexions ($w_{ij}$ et $\beta_j$). Dans la pratique, ces paramètres sont déterminés en se fiant sur une base de données de projets logiciels déjà achevés. Ainsi, on divise la base de données en deux ensembles: un ensemble de projets logiciels pour entraîner le réseau et l'autre pour tester le réseau. L'architecture qui permettra au réseau de neurones de fournir des estimations plus précises sera choisie. Nous présenterons, dans le chapitre 5, les résultats d'un ensemble d'expérimentations que nous avons menées sur le Perceptron multi-couches et le réseau RBFN (*Radial Basis Function Network*) pour étudier la relation entre la précision de leurs estimations et les différents paramètres de leurs architectures.

L'utilisation des réseaux de neurones pour estimer le coût d'un logiciel a trois avantages principaux. Premièrement, ils permettent l'apprentissage automatique à partir des situations et des résultats précédents. L'apprentissage est très important pour les modèles d'estimation des coûts car l'industrie des logiciels est en évolution croissante. Deuxièmement, ils

permettent un traitement parallèle de l'information. Ainsi, le processus d'estimation du modèle profitera de la grande puissance de calcul de toutes les machines disponibles surtout que, souvent, le processus de formulation d'une estimation engage des calculs très intensifs. Troisièmement, ils peuvent modéliser les relations complexes existantes entre la variable dépendante (coût ou effort) et les variables indépendantes du modèle (facteurs de coût). En effet, les réseaux de neurones sont reconnus en tant qu'approximateurs universels de fonctions continues à valeurs dans l'ensemble des réels. Hornik, Stinchcombe et White ont particulièrement montré qu'un réseau de neurones *feedforwad*, avec une seule couche cachée et des fonctions d'activation continues et non-linéaires, peut approcher n'importe quelle fonction continue à valeurs dans l'ensemble des réels et ceci au degré de précision qu'on veut (Hornik, Stinchcombe et White, 1989).

Cependant, l'approche neuronique a trois limitations qui l'empêchent d'être acceptée comme une pratique usuelle en estimation des coûts:

- L'approche neuronique est considérée comme étant une approche *boîte noire*. Par conséquent, il est difficile d'expliquer et d'interpréter son processus.

- Les réseaux de neurones ont été utilisés spécifiquement pour les problèmes de classification et de catégorisation, alors qu'en estimation des coût, nous traitons un problème de généralisation et non pas de classification.

- Il n'existe aucune démarche standard pour le choix des différents paramètres de la topologie d'un réseau de neurones (nombre de couches, nombre d'unités de traitement par couche, valeurs initiales des poids de connexions, etc.).

Dans le chapitre 5, nous étudions la première limitation de l'approche neuronique. En particulier, nous traitons le cas de deux types de réseaux de neurones: le Perceptron multi-couches et le réseau RBFN (Radial Basis Function Network).

### 1.6.2 Modèles d'estimation utilisant les arbres de régression

Les arbres de régression constituent un outil de modélisation simple et de plus en plus répandu par la diversité des applications qui lui font appel. Plusieurs approches de construction de ces arbres ont été développées; ces approches dépendent du type de la connaissance disponible du domaine étudié selon qu'elle est discrète ou continue. Des exemples de telles approches sont CART, ID3 et C4.5 (Breiman et Friedman, 1984; Quinlan, 1986, 1993). La construction de l'arbre de régression s'effectue à partir d'un échantillon de données historiques dit d'apprentissage.

Les arbres de régression, en particulier binaires, ont été utilisés avec succès pour la mise au point de plusieurs modèles d'estimation en génie logiciel (Kitchenham, 1998; Gokhale et Lyu, 1997; Troster et Tian, 1995; Khoshgoftaar, Allen et Deng, 2001; Khoshgoftaar et Seliya, 2002; Pedrycz et Sosnowski, 2001). En estimation des coûts, ils ont été utilisés pour estimer l'effort de développement d'un logiciel (Selby et Porter, 1988; Porter et Selby, 1990a; Porter et Selby, 1990b; Srinivasan et Fisher, 1995). Premièrement, chaque projet logiciel est décrit par un ensemble de variables qui serviront à la prédiction de son effort de développement. Ensuite et progressivement, à partir de la racine de l'arbre, le projet logiciel auquel on veut estimer le coût emprunte le chemin approprié, selon les valeurs de ses variables, en descendant dans l'arborescence jusqu'à l'une des feuilles de l'arbre. Chaque feuille contient une valeur numérique représentant l'effort de développement des projets logiciels associés.

La figure 1.7 illustre un arbre de régression binaire construit à partir des projets du COCOMO'81 (Srinivasan et Fisher, 1995). Dans cet arbre de régression binaire, la racine représente la variable AKDSI (Adjusted Kilo Delivered Source Instructions) de chaque projet logiciel; si la valeur du AKDSI du projet logiciel est supérieure à 315,5, alors son effort est estimé à 9000H-M; sinon, on passe au deuxième niveau de l'arbre où il y a un seul nœud représentant aussi la variable AKDSI et ainsi de suite jusqu'à l'une des feuilles de l'arbre.

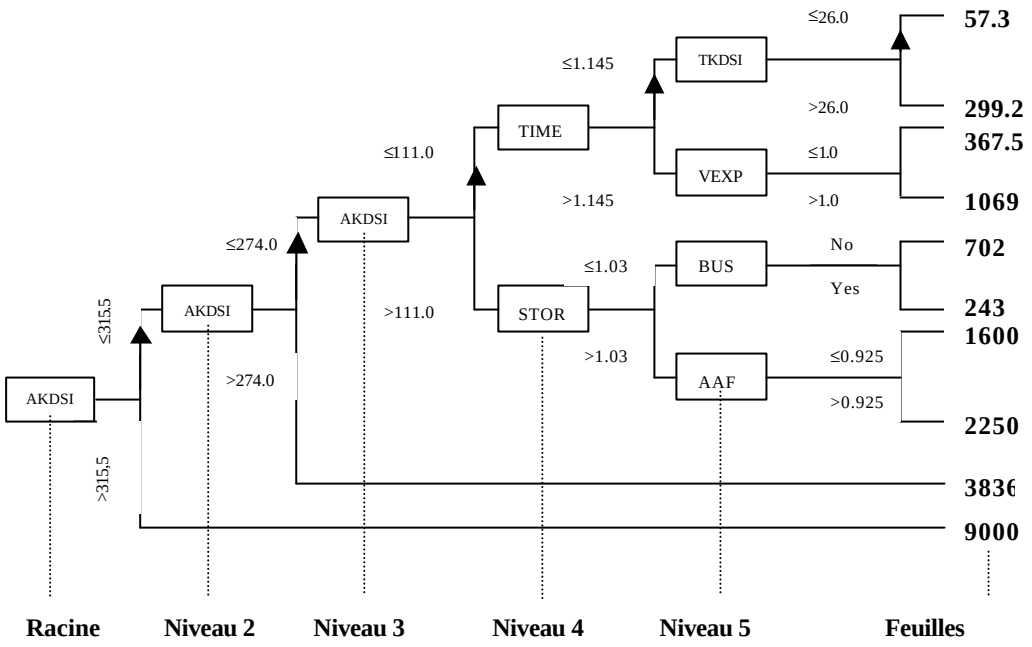



**Figure 1.7** Arbre de régression construit à partir des 63 projets logiciels du COCOMO'81 (Srinivasan et Fisher, 1995).

En général, la plupart des algorithmes de construction d'un arbre de régression binaire, à partir d'un ensemble de projets logiciels historiques, sont basés sur une stratégie descendante. Cette stratégie consiste à choisir, parmi tous les attributs décrivant le projet logiciel, celui qui divise *mieux* l'ensemble de tous les projets logiciels en deux sous-ensembles disjoints. Souvent, on choisit l'attribut qui minimise l'erreur moyenne quadratique (MSE) de la variable dépendante (l'effort de développement) des projets logiciels historiques. L'erreur moyenne quadratique d'un sous-ensemble S de projets logiciels est calculée par:

$$MSE(S) = \frac{\sum_{k \in S} (Effort_k - \overline{Effort})^2}{|S|} \qquad \text{(Équation 1.15)}$$

où $Effort_k$ est l'effort réel du k$^{ème}$ projet logiciel de S et $\overline{Effort}$ est la moyenne arithmétique de $Effort_k$.

Il y a deux cas à distinguer dans le choix de l'attribut qui divise mieux l'ensemble des projets logiciels:

- L'attribut est mesuré par des valeurs discrètes. Dans ce cas, pour chaque attribut $V_i$, on divise l'ensemble de tous les projets logiciels T en des sous-ensembles $T_{ij}$ contenant des projets logiciels ayant la même valeur $A_j$ de l'attribut $V_i$.

- L'attribut est mesuré par des valeurs numériques continues. Dans ce cas, pour chaque attribut $V_i$, on divise l'ensemble de tous les projets logiciels en k-1 sous-ensembles selon l'ordre des valeurs observées de $V_i$ (où k est le nombre de valeurs observées de $V_i$).

Ainsi, on choisit l'attribut $A_i$ qui maximise la différence:

$$?MSE = MSE(T) - \sum_j MSE(T_{ij}) \qquad \text{(Équation 1.16)}$$

L'expansion d'un nœud de l'arbre s'arrête quand le nombre de projets logiciels classifiés dans ce nœud est assez petit ou ces projets logiciels sont jugés similaires. Dans ce cas, le nœud représente une feuille de l'arbre et l'effort estimé de tout nouveau projet logiciel, qui sera classifié dans ce nœud, est calculé en fonction des valeurs réelles des efforts de tous les projets logiciels historiques de ce nœud (en général, on considère la moyenne ou la médiane).

La modélisation de l'estimation des coûts par les arbres de régression présente l'avantage d'être facile à expliquer et à utiliser contrairement au cas de celle utilisant les réseaux de neurones. En effet, un chemin dans l'arbre de régression peut être représenté par une règle du type:

***Si** condition1 et condition2 et … et conditionN **Alors** Effort=CTS*

Par exemple, dans la figure 1.7, le chemin indiqué par des flèches peut être représenté par la règle suivante:

*Si (AKDSI £111.0)et (TIME£ 1.415) et (TKDSI£26.0) alors Effort = 57.3H-H*

Ce genre de règles est facilement interprétable car les règles ressemblent à celles qu'on utilise dans notre quotidien. Ainsi, une modélisation par un arbre de régression binaire n'est qu'un cas particulier d'un système à base de règles Si-Alors.

### 1.6.3 Modèles d'estimation utilisant les algorithmes génétiques

Les algorithmes génétiques sont inspirés des concepts de l'évolution et de la sélection naturelle élaborés par Charles Darwin. Ils ont été utilisés comme des méthodes heuristiques dans les problèmes d'optimisation où l'espace de recherche de la solution est de taille assez grande. Le pionnier du domaine est John Holland qui a le premier tenté d'implanter artificiellement des systèmes évolutifs (Holland, 1975). L'idée de base de la théorie Darwinienne de l'évolution postule que seulement les meilleurs éléments d'une population survivent et transmettent à leurs descendants leurs caractéristiques biologiques à travers des opérations génétiques telles que le *croisement*, la *mutation* et la *sélection*. Dans l'implantation informatique du principe de l'évolution, le problème étudié est représenté par une chaîne binaire (formée de 0 et 1) symbolisant les chromosomes de son équivalent biologique. Ainsi, à partir d'une population, c'est-à-dire à partir d'un ensemble de chaînes binaires (chromosomes),

- les chromosomes les mieux adaptés se reproduisent le plus souvent et contribuent davantage aux populations futures (sélection),
- lors de la reproduction, les chromosomes des parents sont combinés et mélangés pour reproduire les chromosomes des enfants (croisement), et
- le résultat du croisement peut être à son tour modifié par des perturbations aléatoires (mutation).

En général, un algorithme génétique est composé de trois étapes principales bien que d'autres variations sont possibles (Goldberg, 1989; Davis, 1991):

1. Générer aléatoirement une population initiale de N chromosomes (solutions);
2. répéter les sous-étapes suivantes jusqu'à une condition d'arrêt (souvent on fixe un temps d'exécution ou un nombre d'itérations);

2.1. évaluer la *fitness* de chaque chromosome de la population et sélectionner ceux ayant les meilleurs scores;

2.2. reproduire une nouvelle population de chromosomes en appliquant les deux opérations génétiques (croisement et mutation) aux chromosomes sélectionnés dans 2.1;

2.3. remplacer l'ancienne population par la nouvelle population obtenue de 2.1 et 2.2;

3. choisir comme solution au problème celle qui a le meilleur score dans la population finale.

Récemment, les algorithmes génétiques ont été appliqués en estimation des coûts de développement de logiciels (Shukla, 2000; Dolado, 2000; Burgess et Lefley, 2001; Chang, Christensen et Zhang, 2001). Shukla a utilisé les algorithmes génétiques dans l'apprentissage d'un Perceptron multi-couches afin de trouver le vecteur des poids W qui permette au Perceptron multi-couches de fournir des estimations acceptables. Ses expérimentations se sont basées sur la base de données du COCOMO'81. Il identifie un Perceptron multi-couches PN par un vecteur de poids W(PN). À chaque vecteur de poids W(PN), il associe un chromosome de M éléments où M est le nombre d'éléments de W(PM). La figure 1.8 illustre un exemple d'un Perceptron multi-couches ainsi que le chromosome qui lui est associé; ce Perceptron comporte 18 poids de connexions auxquels il associe un chromosome de 18 éléments. Chaque élément du chromosome représente un poids du vecteur W(PN). Pour coder un poids, il lui réserve dix bits. Ainsi, le chromosome associé sera composé de 180 bits. La fonction de *fitness* utilisée pour sélectionner les chromosomes dans une population est:

$$F(W) = \frac{1}{E(W)} \qquad E(W) = \frac{1}{T}\sum_{i=1}^{T}(Effort_{estimé i} - Effort_{réel,i})^2 \qquad \text{(Équation 1.17)}$$

Shukla utilise une technique hybride de sélection des chromosomes qui vont survivre dans la prochaine population. Cette technique considère les deux premiers chromosomes ayant les meilleurs scores F(W) et applique sur les autres la stratégie de la roue roulante dans laquelle chaque chromosome reçoit une part de la roue proportionnelle à son score F(W). Pour l'opérateur de croisement, Shukla a utilisé une version modifiée de l'opérateur standard afin

d'éviter que des bits, représentant des poids appartenant à différentes couches du Perceptron, soient interchangés. Les autres paramètres de l'algorithme, tels que la taille de la population, la probabilité accordée à l'opérateur de croisement ainsi que celle accordée à l'opérateur de mutation, ont été choisis en exécutant plusieurs fois l'algorithme sur les données du COCOMO'81.

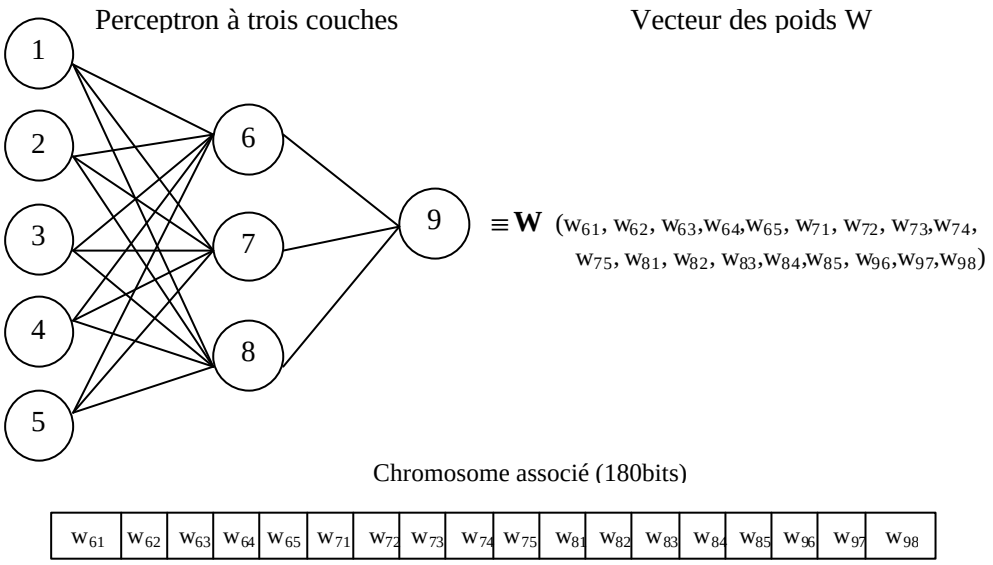



**Figure 1.8** Exemple d'un Perceptron multi-couches et sa représentation avec un chromosome.

Dolado, Burgess et Lefley ont respectivement utilisé la programmation génétique (*Genetic Programing*) pour le développement de modèles d'estimation des coûts de logiciels (Dolado, 2000; Burgess et Lefley, 2001). La programmation génétique (GP) est une variante des algorithmes génétiques qui n'exige pas que la représentation des chromosomes soit seulement sous forme de chaînes binaires, mais elle peut être sous d'autres formes telles que des équations, des programmes, des circuits, et des plans (Koza, 1992). Dans ces deux expérimentations de la programmation génétique en estimation des coûts, l'équation, exprimant l'effort en fonction des variables affectant le coût, a été représentée par un arbre binaire dont les nœuds sont soit des opérandes ou soit des opérateurs. Les opérandes sont les conducteurs du coût tels que la complexité et la fonctionnalité du logiciel, et la compétence du personnel. Ils sont généralement dans les feuilles de l'arbre binaire. Les opérateurs, quant

à eux, représentent les différentes opérations susceptibles d'être dans l'équation exprimant l'effort en fonction des conducteurs du coût. Des exemples de ces opérateurs sont +, -, *, /, Racine carrée, Logarithme, et l'Exponentiel. La figure 1.9 illustre la représentation de l'équation *Effort=0,03453*FP/Ln(FP)* par un arbre binaire:

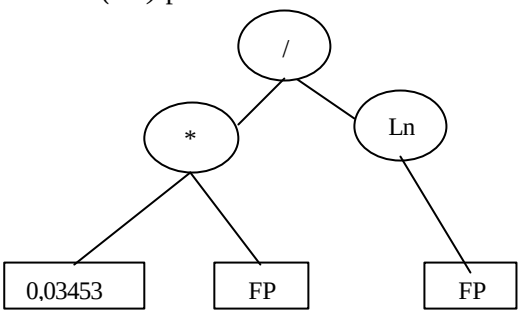



**Figure 1.9** Représentation de l'équation *Effort=0,03453*FP/Ln(FP)* par un arbre binaire.

Les deux variantes de la programmation génétique utilisées respectivement par Dolado, Burgess et Lefley sont presque les mêmes avec des différences mineures au niveau des choix de certains paramètres tels que la fonction d'évaluation de la *fitness* de chaque arbre binaire, la taille de chaque population, et la technique de sélection des arbres binaires. Voici brièvement la description du programme génétique utilisé par Dolado, Burgess et Lefley (Dolado, 2000; Burgess et Lefley, 2001):

1. Générer aléatoirement une population de P équations, c'est-à-dire P arbres binaires. Dolado a utilisé les opérateurs suivants: +, -, *, /, Racine carrée, Carrée, Logarithme, Exponentiel. Burgess et Lefley ont utilisé les opérateurs suivants: +, -, *. Ils ont aussi limité la taille de chaque population à 1 000, la profondeur de chaque arbre binaire à quatre, et le nombre maximal des nœuds dans un arbre à 64.

2. Répéter jusqu'à un nombre maximal de génération (200 pour Dolado et 500 pour Burgess et Lefely);

   2.1. évaluer la *fitness* de chaque équation:

   - Dolado utilise la fonction MSE (*Mean Square Error*):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Effort_{réel,i} - Effort_{estiméi})^2 \qquad \text{(Équation 1.18)}$$

où N est le nombre de projets logiciels, $Effort_{réel,i}$ est l'effort réel du i$^{ème}$ projet, et $Effort_{estimé,i}$ est l'effort estimé du i$^{ème}$ projet par l'équation.

- Burgess et Lefely ont utilisé la moyenne des erreurs relatives (MMRE):

$$MMRE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{Effort_{réel,i} - Effort_{estiméi}}{Effort_{réel,i}}\right| \times 100 \qquad \text{(Équation 1.19)}$$

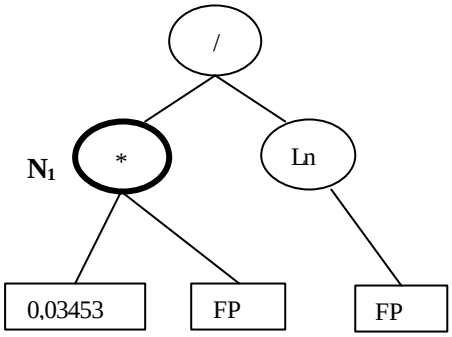
2.2. sélectionner les équations ayant les meilleurs scores de la fonction de *fitness*:

- Dolado utilise la technique de la roue roulante (*Roulette Wheel*) en affectant à chaque équation, i, une probabilité égale à $f_i/Sf_i$ où $f_i$ est la *fitness* du i$^{ème}$ équation.
- Burgess et Lefley choisissent les cinq premières équations ayant les meilleurs scores.

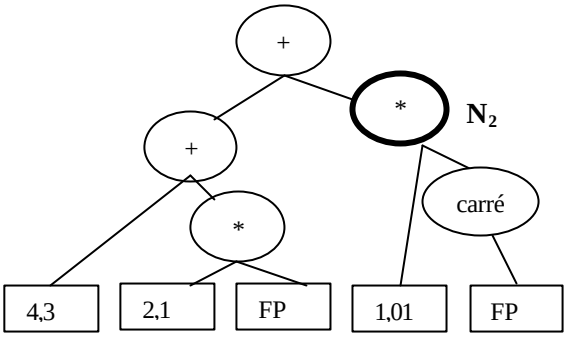
2.3. Appliquer les deux opérateurs de croisement et de mutation aux équations sélectionnées. Le croisement de deux équations consiste au choix d'un nœud $N_1$ dans l'arbre représentant la première équation, d'un nœud $N_2$ dans l'arbre représentant la deuxième équation, et échanger les deux sous-arbres de racine respectivement $N_1$ et $N_2$. La figure 1.10 illustre un exemple de l'opération de croisement appliquée sur les deux équations: *Effort=0,03453*FP/Ln(FP)* et *Effort= 4.3 +2.1*FP+1.01*FP*$^2$. Certaines précautions sont nécessaires pour éviter les problèmes d'incompatibilité des opérateurs avec des opérandes (valeur négative avec la fonction Logarithme par exemple). L'opérateur de mutation est appliqué sur chaque équation sélectionnée en choisissant aléatoirement un nœud de l'arbre binaire (opérateur ou opérande) et en le remplacant par un autre élément (opérateur ou opérande).

2.4. construire la nouvelle population ;

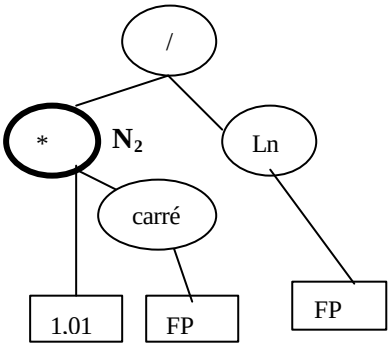
3. choisir l'équation ayant le meilleur score de *fitness*.



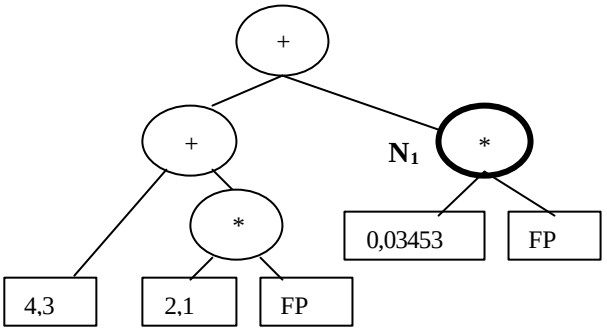

(a)Équation 1: *Effort=0,03453*FP/Ln(FP)*




(b)Équation 2: $Effort=4.3+2.1*FP+1.01*FP^2$




(c)Équation 1.1: $Effort=1,01*Fp^2/Ln(FP)$




(d)Équation 2.1: *Effort=4,3+2,1*FP+0,03453*FP*

**Figure 1.10** Résultats de l'application de l'opérateur de croisement aux deux équations: *Effort=0,03453*FP/Ln(FP)* et $Effort= 4.3 +2.1*FP+1.01*FP^2$.

L'utilisation du principe de l'évolution en estimation des coûts présente l'avantage de fournir une recherche heuristique de l'équation exprimant l'effort en fonction des conducteurs du coût. Ainsi, à partir d'un ensemble d'équations initiales, l'algorithme recherche celle qui représente adéquatement la relation. Ceci présente un avantage sur les techniques paramétriques classiques telles que la régression linéaire où une seule forme de l'équation est évaluée. Cependant, comme dans le cas des réseaux de neurones, la configuration d'un algorithme génétique (ou un programme génétique) nécessite le choix de plusieurs paramètres tels que la fonction de *fitness*, la taille de chaque population et la taille de l'arbre binaire représentant l'équation. Ces paramètres sont souvent déterminés par expérimentations. Ils dépendent donc de la base de projets logiciels utilisée. Aussi, nous pouvons adresser aux algorithmes génétiques la même critique que celle mentionnée dans le

cas des réseaux de neurones au sujet de l'interprétation et de la compréhension du processus d'un algorithme génétique.

## 1.7 LA VALIDATION DES MODÈLES D'ESTIMATION DES COÜTS

Dans la littérature, on retrouve un ensemble de critères de validation de n'importe quel modèle d'estimation. Ces critères représentent les objectifs du modèle:

- Précision: c'est l'objectif principal de n'importe quel modèle d'estimation. En effet, le rôle du modèle est de fournir des estimations précises (à un degré acceptable) aux coûts des nouveaux projets. Ainsi, pour chaque nouveau projet logiciel, le coût estimé doit être proche de son coût réel.

- Constructive: le modèle doit être facilement interprétable afin qu'on puisse, quand il est nécessaire, expliquer son processus d'estimation. Ceci permettra aux estimateurs de découvrir d'autres règles (connaissances) d'estimation des coûts qui n'étaient pas connues. Ainsi, le modèle pourra être considéré comme un fournisseur, en plus des estimations, de nouvelles connaissances dans le domaine d'estimation des coûts (*Knowledge Discovery*).

- Stable: le modèle doit fournir des estimations assez proches quand ses entrées (valeurs des facteurs influençant le coût) sont de même. Il doit donc éviter les discontinuités dans l'évaluation des coûts. Ce cas peut se présenter quand certaines des entrées du modèle sont binaires.

- Opérationnel: le modèle doit être facile à utiliser. Ainsi, il doit utiliser le minimum de facteurs ; ses options doivent être facile à configurer, et il doit être efficace (consomme peu de ressources en termes de temps d'exécution et d'espace mémoire).

Le premier critère de la précision d'un modèle est celui qui a été le plus débattu dans la littérature. Ainsi, nous présentons un aperçu sur les résultats des différents travaux de recherche ayant comme objectif l'évaluation de la précision d'un modèle d'estimation des

coûts. Les autres critères d'évaluation (constructive, stable et opérationnel) seront traités au fur et à mesure que nous présenterons notre modèle d'estimation des coûts (Chap. 4 et 5).

Le souci principal des chercheurs et des industriels, quand ils expérimentaient les modèles d'estimation des coûts, était souvent celui des performances au niveau de la précision des estimations qu'ils fournissent. Ainsi, dans la littérature, de nombreuses études empiriques ont été menées pour évaluer la précision des estimations de plusieurs modèles paramétriques et/ou non-paramétriques décrits dans les deux sections précédentes. En général, ces études empiriques consistent, premièrement, à la mise au point de modèles d'estimation à partir d'une (ou plusieurs) base de projets logiciels déjà achevés et, deuxièmement, à l'évaluation de la performance de ces modèles sur la même (ou une autre) base de projets logiciels. Souvent, les indicateurs utilisés pour évaluer la précision d'un modèle d'estimation sont basés sur la comparaison des valeurs estimées avec les valeurs réelles des coûts des projets logiciels:

- La valeur absolue de l'erreur relative d'une estimation (*Magnitude Relative Error*) MRE définit par:

$$MRE = \left| \frac{Effort_{réel} - Effort_{estimé}}{Effort_{reél}} \right| \qquad \text{(Équation 1.20)}$$

  Autant la valeur du MRE d'une estimation est petite autant l'estimation est précise. Souvent, dans la littérature, on considère une estimation ayant une MRE inférieure ou égale à 0,25 comme étant acceptable (Conte, Dunsmore et Shen, 1986; Boehm, 1981). Miyazaki et al. ont proposé une autre définition de MRE vu que celle ci-dessus pénalise plus les sur-estimations que les sous-estimations (Miyazaki et al, 1991):

$$MRE = \left| \frac{Effort_{réel} - Effort_{estimé}}{min(Effort_{reél}, Effort_{estimé})} \right| \qquad \text{(Équation 1.21)}$$

- La moyenne des erreurs relatives, MMRE (Équation 1.19);

- le pourcentage de projets logiciels ayant une MRE inférieure ou égale à une valeur p (souvent p=0,25), Pred(p):

$$Pred(p) = \frac{K}{N} \qquad \text{(Équation 1.22)}$$

où K est le nombre de projets logiciels ayant une MRE inférieure ou égale à p, N est le nombre total de projets logiciels de test. Dans la littérature, un modèle ayant une valeur de Pred(0,25) égale à 70% est dit acceptable (Conte, Dunsmore et Shen, 1986).

- Le coefficient de corrélation entre les valeurs estimées et les valeurs réelles des coûts ;

- Dans le cas des modèles linéaires, le coefficient de détermination $R^2$.

La plupart des études empiriques ont montré qu'aucune technique n'a pu prouver qu'elle performe mieux que les autres dans toutes les situations (Tab. 1.1). Par exemple, Shepperd et Schofield, Niessink et Van Vliet ont trouvé que la précision de la technique d'estimation par analogie est meilleure que celle de la technique de régression linéaire; par contraste, Briand, Langley et Wieczorek, Myrtveit et Stensrud ont rapporté le contraire (Shepperd et Schofield, 1997; Niessink et Van Vliet, 1997; Briand, Langely, et Wieczorek, 2000; Myrtveit et Stensrud, 1999). En effet, la performance d'un modèle d'estimation dépend, au moins, de trois facteurs:

- La technique utilisée pour la mise au point du modèle. En effet, les techniques paramétriques exigent la connaissance de la forme de la relation exprimant l'effort en fonction des conducteurs du coût. Ainsi, si la forme adoptée ne représente pas convenablement la réalité, le modèle risque de fournir des estimations moins précises. Dans le cas des modèles non-paramétriques, l'idée consiste à trouver une approximation à la forme réelle de la relation. Ainsi, s'il s'avère que cette relation a une forme bien déterminée telles que linéaire ou analytique, les modèles paramétriques ont plus de chances d'être très précis que les modèles non-paramétriques.

- Les caractéristiques de la base des projets logiciels à partir de laquelle le modèle a été mis au point. En effet, les paramètres d'un modèle d'estimation (paramétrique ou non-paramétrique) se définissent, en général, en fonction de ces projets logiciels. Des exemples de caractéristiques dont les impacts sur la performance d'un modèle ont été étudiés dans la littérature sont (Shepperd et Kadoda, 2001):

  a) la taille de la base de projets logiciels, c'est-à-dire le nombre de projets logiciels qu'elle contient. Souvent un grand nombre de projets logiciels permettent au modèle de mieux représenter son environnement. En effet, Shepperd et Kadoda ont étudié l'impact du nombre de projets sur la précision de trois techniques d'estimation: régression *stepwise*, CBR et règles d'induction; ils ont observé que plus le nombre de projets est élevé plus la précision du modèle est élevé (Tab. 1.1b). Cette observation est aussi vérifiée dans le cas d'un modèle d'estimation utilisant les réseaux de neurones, dont le processus d'apprentissage nécessite un grand nombre de projets logiciels afin de permettre au modèle de mieux généraliser (Srinivasan et Fisher, 1995; Wittig et finnie, 1997; Serluca 2000). Nous étudions l'impact de cette caractéristique sur la performance d'un Perceptron multi-couches adoptant l'algorithme de rétro-propagation (*Backpropagation*) dans l'apprentissage (voir Chap. 5).

  b) Le nombre d'attributs décrivant les projets logiciels. Souvent, dans la littérature un modèle d'estimation ne considère que les attributs ayant des données réelles disponibles, comme étant les principaux conducteurs du coût. Il exclut donc des attributs qui n'ont pas de données enregistrées bien qu'ils puissent être significatifs pour l'estimation des coûts. Aussi, la sélection d'un sous-ensemble d'attributs qui décrit mieux un projet logiciel représente une tâche très ardue dans le processus d'estimation des coûts. En général, il y a deux critères qu'un attribut candidat doit satisfaire: 1) l'attribut doit être significatif pour l'estimation des coûts; 2) l'attribut doit être indépendant de tous les autres attributs (Boehm, 1981; Mayhrauser, 1990). Dans le chapitre 4, nous étudierons, en particulier, l'impact du nombre

d'attributs considérés sur la précision d'un modèle d'estimation utilisant le raisonnement à base de cas (CBR).

c) La présence des points extrêmes (*outliers*). C'est le cas de la régression linéaire où ces points affectent largement les valeurs des paramètres du modèle (Tab. 1.1b).

- Les caractéristiques de la base de projets logiciels utilisés dans l'évaluation de la performance du modèle. En effet, il est peu probable qu'un modèle fournisse des estimations précises pour des projets logiciels qui diffèrent beaucoup de ceux de la base de projets historiques. Ainsi, Miyazaki et Mori ont testé le modèle COCOMO'81 sur des projets logiciels relatifs à l'environnement japonais. Ils ont observé des imprécisions énormes qui ont été rectifiées après la calibration du COCOMO'81 à l'environnement japonais (Miyazaki et Mori, 1985). Aussi, Kemerer avait obtenu des estimations imprécises en appliquant le modèle COCOMO'81 sur des projets logiciels d'applications de gestion (Kemerer, 1987). En effet, la base de données COCOMO'81 ne contient que cinq applications de gestion parmi 63 dont la plupart sont des applications scientifiques. Par contraste, Briand et al. ont évalué quatre modèles d'estimation dans deux expérimentations différentes sur une base de données de 63 projets logiciels du domaine de la banque. Dans la première expérimentation, ils ont mis au point les quatre modèles en utilisant les mêmes 63 projets logiciels. Dans la deuxième expérimentation, ils ont mis au point les quatre modèles en utilisant 143 autres projets logiciels du domaine des assurances, du *manufacturing*, et de l'administration publique. Les résultats obtenus de ces performances de quatre modèles dans les deux expérimentations étaient similaires (Briand et al, 1999). Briand et al. expliquent ceci par le fait que les 143 autant que les 63 projets logiciels étaient décrits avec presque les mêmes attributs (taille, type de l'application et la plate-forme utilisée) et que les 63 projets du domaine de la banque sont similaires aux 143 des autres domaines. En général, afin d'éviter les effets des différences entre les projets logiciels ayant été utilisés pour la mise au point du modèle et ceux qui seront utilisés pour évaluer la précision du modèle, la pratique consiste à diviser un

ensemble homogène de projets logiciels en deux sous-ensembles: le premier servira pour la mise au point du modèle et le deuxième pour le test du modèle.

**Tableau 1.1**
Résumé des résultats de comparaison des performances de plusieurs modèles d'estimation des coûts.

(a) Référence: Shepperd et Schofield, 1997

| Base de données | CBR MMRE ( %) | Régression linéaire MMRE (%) |
|---|---|---|
| Albrecht | 62 | 90 |
| Atkinson | 39 | 45 |
| Kemerer | 62 | 107 |
| Mermaid | 78 | 252 |
| Desharnais | 64 | 66 |

(b) Référence: Shepperd et Kadoda, 2001
Base de données: Albrecht, Atkinson, Desharnais , Finnish, Mermaid

| | 20 projets logiciels | | 100 projets logiciels | |
|---|---|---|---|---|
| | Sans points extrêmes | Avec points extrêmes | Sans points extrêmes | Avec points extrêmes |
| Régression *Stepwise* MMRE (%) | 484,59 | 313,54 | 404,95 | 258,12 |
| Arbres de décision MMRE (%) | 197,53 | 162,12 | 106,80 | 187,70 |
| CBR MMRE (%) | 198,01 | 245,99 | 245,99 | 166,38 |

(c) Référence: Srinivasan et Fisher, 1995
Base de données: Boehm'81 et Kemerer

| | MMRE (%) | $R^2$ |
|---|---|---|
| Arbre de décision (CARTX) | 364 | 0,83 |
| Réseau de neurones avec Rétro-propagation | 70 | 0,80 |

(d) Référence: Briand et al., 1999
Base de données: 206 applications de gestion de la Finlande

| | Régression *stepwise* | ANOVA[1] *stepwise* | CART | CBR |
|---|---|---|---|---|
| MMRE | 0,53 | 0,567 | 0,52 | 1,16 |
| MdMRE | 0,44 | 0,446 | 0,39 | 0,61 |
| Pred(25) % | 31 | 34 | 34 | 24 |

[1] ANOVA *stepwise* est une méthode qui utilise l'analyse de variance pour les conducteurs du coût mesurés sur une échelle au plus Ordinale et la régression linéaire classique pour les conducteurs du coût mesurés sur une échelle intervalle ou ratio (Kitchenham, 1998) .

(e) Référence: Burgess et Lefley, 2001
Base de données: Desharnais

| | MMRE (%) | Pred(25) % |
|---|---|---|
| Programmation génétique | 44,55 | 23,3 |
| Réseau de neurones | 60,36 | 55,56 |
| Régression linéaire | 46,18 | 55,56 |

(f) Référence: Dolado, 2000

| | Abran et Robillard | Albrecht | Boehm'81 | Desharnais |
|---|---|---|---|---|
| Régression linéaire | MMRE=23,64 | 53,13 | 113,36 | 54,28 |
| | Pred(25)=57,14 | 54,17 | 17,46 | 44,26 |
| Programmation génétique | MMRE=25,0 | 54,8 | 17,8 | 62,3 |
| | Pred(25)=77,3 | 64 | 15,6 | 51,6 |

(j) Référence: Jeffery, Ruhe et Wieczorek, 2001
Base de données: ISBSG

| | CBR | Régression linéaire | Régression linéaire robuste | CARTX | ANOVA *stepwise* |
|---|---|---|---|---|---|
| MMRE (%) | 114,5 | 89,5 | 84,8 | 156 | 130 |
| Pred(25) % | 17 | 00 | 00 | 08 | 00 |

(h) Référence: Mendes et Counsell, 2002
Base de données: 37 applications web hypermédia

| | MMRE | MdMRE | Pred(25) % |
|---|---|---|---|
| CBR | 0,12 | 0,09 | 90,91 |
| Régression linéaire | 0,03 | 0,03 | 100 |
| CART | 0,22 | 0,11 | 81,82 |

## 1.8 CONCLUSION

L'estimation des coûts de développement de logiciels est l'un des problèmes critiques les plus débattus en métrologie de logiciels. La maîtrise et l'évaluation de ces coûts semblent encore préoccuper aussi bien les chercheurs que les responsables et les chefs de projets logiciels. Dans ce chapitre, nous avons présenté un ensemble de techniques d'estimation des coûts, spécifiquement celles se basant sur la modélisation. Ainsi, nous avons décrit des exemples de modèles paramétriques et non-paramétriques. Chaque type de modèle d'estimation a des avantages et des inconvénients et aucun modèle n'a pu prouver qu'il performe mieux que tous les autres dans toutes les situations.

L'objectif de cette synthèse sur les différentes techniques d'estimation des coûts est d'identifier celle qui pourra intégrer facilement nos trois critères d'intelligence, à savoir la tolérance des imprécisions, la gestion des incertitudes au niveau des valeurs estimées et l'apprentissage. Les techniques non-paramétriques semblent offrir plus de flexibilité, du fait qu'elles utilisent des approches de l'Intelligence artificielle, pour le développement de notre modèle d'estimation dit *intelligent*. Spécifiquement, nous optons pour la technique non-paramétrique basée sur un raisonnement par analogie (*Case-Based Reasoning*). L'intégration de nos trois critères d'intelligence dans la technique d'estimation des coûts par analogie fera l'objet des chapitres 4 et 5. Cependant, nous présentons dans le prochain chapitre les concepts et les outils nécessaires pour intégrer ces trois critères d'intelligence dans notre modèle.

# CHAPITRE II

# INTELLIGENCE ARTIFICIELLE ET LOGIQUE FLOUE

L'intégration de nos trois critères d'intelligence dans l'estimation des coûts par analogie nécessite l'utilisation des concepts et des méthodes de l'Intelligence artificielle. Depuis son apparition dans les années 50, l'Intelligence Artificielle (IA) a été l'objectif de plusieurs travaux de recherche émanant de plusieurs communautés scientifiques: philosophes, psychologues, informaticiens, mathématiciens, etc. Son objectif est de développer des systèmes *intelligents* capables d'imiter certaines capacités des humains reconnues par tous comme étant des caractéristiques fondamentales de l'intelligence. Cependant, les approches de l'Intelligence artificielle, classifiées principalement dans deux catégories: Symbolisme et Connexionnisme, ont développé séparément des stratégies et des techniques qui ont été plus au moins utilisées avec succès dans le développement de ces systèmes intelligents.

Récemment, le domaine de l'intelligence artificielle a connu des progressions remarquables suite au développement de plusieurs théories solidement fondées telles que la Logique Floue (FL), les Algorithmes Génétiques (GA) et la théorie du Chaos. La Logique Floue, développée par Zadeh en 1965, est celle qui a contribué, peut être, le plus dans le développement de l'Intelligence Artificielle moderne. Ce chapitre présente donc les concepts de base de la Logique Floue ainsi que son rôle dans le développement des systèmes intelligents. Ce sont ces concepts que nous utiliserons principalement pour l'intégration de nos trois critères d'intelligence dans la technique d'estimation par analogie.

## 2.1 INTRODUCTION

L'intelligence est une caractéristique des humains et elle est considérée comme la principale qui nous distingue des autres êtres vivants. Les philosophes ont essayé, il y a longtemps, de comprendre et de répondre à deux questions relatives au cerveau humain (organe responsable de notre intelligence):

1- Comment le cerveau humain fonctionne-t-il ?
2- Comment peut-on faire des cerveaux aux non-humains ?

Cependant, le concept d'*intelligence* n'est pas communément bien défini. Comme d'autres concepts tels que la *mort*, l'*existence* et l'*unicité*, il est de nature philosophique. Il n'est donc pas surprenant de trouver dans la littérature plusieurs définitions de ce concept. Voici deux définitions données dans le *Essential English Dictionary*, Collins, Londres, 1990:

> «Someone's intelligence is their ability to understand and learn things»
> «Intelligence is the ability to think and understand instead of doing things by instinct or automatically»

En général, nous pouvons dire que le concept d'*intelligence* regroupe plusieurs caractéristiques parmi lesquelles nous pouvons citer l'apprentissage, la créativité, la facilité de comprendre et la gestion des incertitudes. Ces caractéristiques ne sont pas présentes avec les mêmes degrés chez les humains. On retrouve, par exemple, que certains sont considérés très *intelligents* en mathématiques contrairement à ce qu'ils sont en géographie; d'autres sont très *intelligents* en investissement d'argent à l'opposé de certains qui sont très *intelligents* en gaspillage d'argent. Cependant, tous les humains sont capables d'apprendre et de comprendre. La différence est que cette compétence dépend des domaines.

En estimation des coûts, nous identifions au moins trois critères qu'un modèle d'estimation doit satisfaire, en plus de la précision de ses estimations, pour être considéré comme un modèle intelligent: (1) il doit tolérer les imprécisions tout au long du processus de formulation d'une estimation; (2) il doit gérer les incertitudes au niveau des estimations fournies; et (3) il doit être capable d'apprendre.

Ce chapitre présente les concepts de base que nous utiliserons pour intégrer les trois critères d'intelligence, cités ci-dessus, dans notre modèle d'estimation des coûts. Tous ces concepts émanent du domaine de l'intelligence artificielle, spécifiquement celui de la logique floue. Ainsi, nous présentons, dans la première section, un aperçu général sur l'intelligence artificielle; en particulier, nous survolons les principes des deux approches: Symbolisme et Connexionnisme. Dans la deuxième section, nous discutons du rôle et la place de la logique floue dans le développement des systèmes intelligents. Ensuite, nous présentons les concepts de base de la logique floue tels que les ensembles flous, les variables linguistiques et le raisonnement flou. Ces concepts seront utilisés ultérieurement dans les chapitres 3, 4 et 5.

## 2.2 INTELLIGENCE ARTIFICIELLE: VUE D'ENSEMBLE

L'expression *intelligence artificielle* est apparue officiellement lors de la conférence du Dartmouth College en 1956. Cependant, ses origines ne peuvent être datées exactement ou reliées à une référence bien précise. Selon Chaudet et Pellering, sa renaissance date de 1900 quand David Hilbert présenta une liste de 33 problèmes théoriques qui, d'après lui, attireraient les mathématiciens du siècle à venir. Le 33ème de ces problèmes était celui de la décidabilité, c'est-à-dire l'existence d'une procédure générale permettant d'évaluer une proposition logique donnée à une des valeurs de vérité: *vraie*, *fausse* ou autre. La première preuve de l'impossibilité de construire une telle procédure revient à Kurt Gödel qui en 1931 démontra que, dans tout système axiomatique manipulant l'arithmétique des nombres naturels, il existe des propositions pour lesquelles le système est incapable d'assigner la valeur *vraie* ou *fausse* (Chaudet et Pellegrin, 1998). La deuxième réponse, attribuée à Alan Turing en 1936, parle de la parenté entre calculabilité et décidabilité. Turing utilisa la calculabilité pour reproduire le résultat de Gödel sur l'indécidabilité de certaines propositions logiques. En effet, Turing démontra l'existence de fonctions non calculables et par conséquent, leurs résultats, qui sont des propositions logiques, sont indécidables (Turing, 1936).

En 1950, Turing esquisse une théorie de l'esprit fondée sur les concepts de *Test de Turing* et de la *Machine universelle de Turing* introduits respectivement dans ses deux articles de 1950 et 1936. (Turing, 1936, 1950). Elle inspire aux pionniers de l'intelligence artificielle deux programmes (Jordan et Russell, 1999):

- L'intelligence artificielle *faible* (une discipline d'ingénierie), étudiée par certains groupes du MIT (McCarthy, Minsky et al.), a comme objectif le développement des systèmes intelligents simulant certains des comportements humains admis et reconnus par tous comme étant intelligents. Cette position évite de se lancer dans une recherche difficile (et hasardeuse) sur les processus cognitifs en jeu. Cette approche domine actuellement le domaine de l'intelligence artificielle moderne ainsi que celui des sciences cognitives.

- L'intelligence artificielle *forte* étudiée à Carnegie-Mellon (Newell et Simon) a comme objectif de modéliser l'esprit humain en construisant un programme qui, conjointement avec l'ordinateur, reproduira toute la cognition humaine.

Jordan et Russell remarquent que, de plus en plus, les différences entre les modèles de l'intelligence émanant de l'ingénierie et ceux inspirés des modèles cognitifs sont très négligeables comparées au fossé qui existe encore entre les performances de ces modèles et celles de l'intelligence des humains (Jordan et Russell, 1999). En plus, ces modèles postulent, en général, que la cognition humaine n'est qu'une forme de *computation* et, par conséquent, elle peut être réalisée par un ordinateur. Les approches computationelles de la cognition peuvent être regroupées en deux catégories: l'approche *symbolique* et l'approche *connexionniste*. Nous présentons brièvement, dans cette section, les principaux concepts de ces deux approches.

Issue des travaux de Newell et Simon, l'approche symboliste assume, comme hypothèse de travail, qu'il existe une analogie de fonctionnement entre le cerveau humain et l'ordinateur (Newell et Simon, 1972; Newell, 1980, 1990; Pylyshyn, 1984, 1989). Elle postule qu'un système intelligent est un système physique de traitement de symboles, lequel opère à partir de règles. C'est ici qu'on retrouve les fameuses thèses comme:

*Cognition = représentation + computation,*
*Esprit = programme*, et *cerveau = ordinateur*.

Le symbolisme avait longtemps dominé les sciences cognitives. Quelques avancées et certaines réalisations étaient nées pour souligner le succès de cette approche (les systèmes experts, par exemple). Cependant, la nature symbolique et sérielle de l'approche symboliste ne peut traiter des problèmes complexes vu l'explosion combinatoire qu'elle peut manifester. Pour pallier ce type de problèmes, l'approche connexionniste, dont la métaphore est neurobiologique plutôt que logico-mathématique, propose une méthode massivement parallèle au traitement de l'information, basée sur les opérations d'un réseau neuronal composé de plusieurs unités simples et interconnectées (Rumelhart et McClelland, 1986; Rumelhart, 1989; Smolensky, 1988; Proulx, 1994). Le connexionnisme considère la cognition comme le résultat d'une interaction globale entre des parties élémentaires

(neurones) d'un système. En effet, la neurobiologie avait découvert que le cerveau humain était constitué d'un ensemble de régions et que chacune est responsable d'une ou plusieurs fonctionnalités; aussi l'interaction entre ces différentes régions produit d'autres types de fonctionnalités (comportements mentaux). De ce fait, l'approche connexionniste va opter pour une approche computationnelle similaire à celle du cerveau humain (*Brain-like computation*) afin d'aboutir à ce qu'on appelle la machine cognitive.

Depuis, les deux approches (Symboliste et Connexionniste) s'affrontent et débattent des questions philosophiques et scientifiques. Le tableau 2.1 résume les grands thèmes de débat entre les deux approches. La question principale ici est: le connexionnisme est-il un complément ou une alternative au symbolisme? De plus en plus, les sciences cognitives vont vers l'adoption d'une approche hybride déléguant par exemple la perception à une couche connexionniste et le raisonnement à une couche symbolique (Jordan et Russell, 1999). L'idée est de profiter de la rigueur de l'approche symboliste et de la flexibilité et de l'apprentissage de l'approche connexionniste. Un défi pour les sciences cognitives est de trouver une bonne passerelle entre ces deux approches.

Au niveau de l'intelligence artificielle *faible*, plusieurs travaux de recherche ont déjà montré l'utilité et l'efficacité d'une approche hybride plutôt qu'une approche purement symboliste ou connexionniste dans la résolution de plusieurs problèmes complexes (Tirri, 1991; Towell, 1992; Medsker, 1994; Jang, Sun et Mizutani, 1997). Dans ce travail de recherche, nous adoptons aussi une approche hybride pour le développement de notre modèle intelligent d'estimation des coûts de logiciels. En effet, nous utilisons une panoplie de techniques relevant de l'approche symboliste et connexionniste des sciences cognitives. Des exemples de ces techniques sont la théorie des ensembles flous, le raisonnement par analogie, la théorie des possibilités et les réseaux de neurones.

**Tableau 2.1**
Thèmes de débat entre l'approche symboliste et l'approche connexionniste.

| Symbolisme | Connexionisme |
|---|---|
| 1- Il est basé sur la métaphore logico-mathématique de l'ordinateur. | 1- Il est basé sur la métaphore neurobiologique du cerveau humain. |
| 2- La cognition est la manipulation de symboles élémentaires fondée sur leur forme par des opérateurs. C'est ici que se trouvent les fameuses thèses controversées:<br>- Cognition=représentation+computation<br>- Cerveau=ordinateur, et<br>- Esprit=programme. | 2- La cognition est le résultat d'une interaction globale entre les différents modules composant le système. Exactement comme dans le cas du cerveau humain, le système est composé de plusieurs modules contenant des unités élémentaires dont les propriétés sont semblables à celles des neurones. |
| 3- Il adopte un traitement séquentiel de l'information semblable à celui de l'ordinateur classique (*serial computation*) | 3- Il adopte un traitement parallèle de l'information semblable à celui du cerveau humain (*Brain-style computation*) |
| 4- On doit décrire explicitement dans un système symboliste toutes les séquences de toutes les actions dans toutes les circonstances du domaine modélisé (construire pour construire la tâche). | 3- On peut commencer par un système à zéro connaissance et au fur à mesure qu'il interagit avec son environnement il se configure; ainsi de nouvelles connaissances émergent dans le système (construire le processus et non pas la tâche). |
| 5- Le processus est clair et rigoureux mais très fragile devant des situations qui n'ont pas été prévues à l'avance (cas des systèmes experts). | 5- Le processus est très robuste mais 'boîte noire'. Ainsi au fur à mesure de son interaction avec l'environnement, de nouvelles connaissances émergent sans savoir comment. |
| 6- Le processus ne donne que peu d'intérêt à l'apprentissage tandis qu'il consacre tous les efforts à la modélisation des représentations et aux tâches d'inférence pour le *problem solving* | 6- Le processus considère l'apprentissage en tant qu'une priorité indispensable pour son adaptation et son utilité. |
| 7- Du fait de son traitement séquentiel du problème, un système symboliste est inadéquat pour des problèmes du type *constraints-satisfaction problems*. | 7- Du fait de son traitement parallèle, un système connexionniste est adéquat pour la résolution des problèmes de type: *constraints-satisfaction problèms*. |
| 8- Du fait de son traitement séquentiel du problème, un système symboliste n'est pas tolérant aux fautes. S'il y a une erreur dans la phase de représentation ou de *computation*, le processus risque de se bloquer et ne donner aucune réponse. | 8- Du fait de son traitement parallèle, un système connexionniste est tolérant aux fautes: certains modules peuvent être endommagés sans que cela ne bloque carrément le fonctionnement du processus. On aura juste une dégradation des fonctionnalités. |

## 2.3 LA PLACE DE LA LOGIQUE FLOUE DANS L'INTELLIGENCE ARTIFICIELLE

Depuis que Alan Turing a publié son monumental article *Computing Machinery and Intelligence* (Turing, 1950), les machines, spécifiquement les ordinateurs, ont été l'objet de plusieurs travaux de recherche ayant comme objectif la simulation de certaines capacités du cerveau telles que l'apprentissage, la gestion des incertitudes et l'adaptation[2] (Negnevitsky, 2001). Bien que la plupart des fondateurs de l'intelligence artificielle estimaient qu'en l'an 2000 les ordinateurs auraient une intelligence similaire à celle des humains, jusqu'à présent on est encore loin d'atteindre cet objectif. En effet, les questions provenant du *dark ages* de l'intelligence artificielle telles que: les machines peuvent-elles penser? et les machines peuvent-elle reproduire toute l'intelligence humaine? sont encore sans réponse. Aussi, plusieurs problèmes relevant de notre vie quotidienne, tels que garer un véhicule dans un stationnement ou déplacer un robot comme les humains, sont encore sans solution satisfaisante.

Aujourd'hui, un des grands défis pour l'intelligence artificielle se trouve dans les domaines où les connaissances sont souvent *imprécises* et *incertaines*. Dans nos sociétés modernes on a besoin, de plus en plus, de produits intelligents facilitant notre vie quotidienne: machines à laver, caméras, magnétoscopes, téléviseurs, etc. Dans ces produits, on exige un haut niveau de ce qu'on appelle le MIQ (*Machine Intelligent Quotient*). Un exemple typique sont les caméras modernes dotées d'un mécanisme intelligent permettant d'ajuster et d'adapter automatiquement la photo dépendamment de votre position; par conséquent, le rôle du photographe est facultatif contrairement aux caméras classiques.

Cependant, du fait que les approches classiques de l'intelligence artificielle ne manipulent que les deux valeurs de vérité: *vraie* et *fausse* (le principe du *excluded middle* de la logique d'Aristote), leur application à plusieurs domaines, où la connaissance était imprécise et incertaine, est restée très limitée. L'importance de la manipulation des imprécisions et des incertitudes découle du fait que nos connaissances sont souvent incomplètes et partielles. La

[2] On citera comme exemples les deux fameux programmes: *Advice Taker* de McCarthy et le *General Problem Solver* (GPS) de Newell et Simon. Ces deux programmes prétendaient développer une méthode générale dite *intelligente* pour la résolution de tous les problèmes.

*partialité* plutôt que la *totalité* est omniprésente dans notre quotidien. Généralement, notre connaissance est *partielle*, notre croyance est *partielle*, notre compréhension est *partielle*, et la vérité elle aussi est *partielle*. Ce dernier concept de partialité dans la vérité, est à l'origine de toute la logique floue. Le débat sur les valeurs possibles de vérité dépasse le cadre de l'intelligence artificielle et on le retrouve aussi dans les religions, les philosophies, la politique et des autres domaines où l'humain joue un rôle primordial. Il est peut-être impossible de le dater ou de le relier à une référence bien précise.

Bien que les théories scientifiques classiques, y compris l'intelligence artificielle, qui se basent sur la précision et la non- ambiguïté (deux valeurs de vérité *vraie* et *fausse*), aient pu obtenir des succès considérables comme l'arrivée des humains à la lune, la construction d'ordinateurs très puissants et l'invention de télescopes explorant des régions lointaines dans notre univers, n'ont pas encore réussi à construire des robots qui se déplacent de la même façon que les animaux ou les humains, ni à automatiser la conduite de voitures dans le trafic routier ni à développer des outils de traduction d'un langage vers un autre sans l'intervention des humains. Selon Lotfi Zadeh, le fondateur de la logique floue, les limites des théories classiques viennent du fait qu'elles exigent et manipulent seulement des informations précises (Zadeh, 1997).

Introduites en 1965 par Lotfi Zadeh, la logique floue est longtemps restée marginale, n'intéressant qu'un cercle restreint d'adeptes parmi les enseignants et les chercheurs (Zadeh, 1965). Cette traversée du désert que connaissent la plupart des nouvelles disciplines (du fait que très souvent elles basculent l'ordre établi) a duré au moins une dizaine d'années. Mais la logique floue va connaître, au terme de cette période de gestation, un essor prodigieux grâce à l'évolution de l'intelligence artificielle et aux applications pratiques dans le domaine de la commande et du contrôle des automatismes de processus.

La logique floue propose des modes de raisonnement approximatifs plutôt qu'exacts. C'est principalement le mode de raisonnement utilisé dans la plupart des cas par les humains. Par conséquent, il semblait au début, que cette nouvelle science (logique floue) serait la bienvenue au sein de la communauté de l'intelligence artificielle. La réalité c'est que la

plupart l'ont reçu avec un scepticisme et parfois même avec hostilité. Il y a plusieurs raisons derrière cette réticence:

- Les fondateurs de l'intelligence artificielle ne manipulaient auparavant que deux valeurs de vérité: *vraie* et *fausse.*
- Une mauvaise compréhension, de la part de la communauté de l'intelligence artificielle, de ce que la logique floue pouvait offrir comme techniques afin de résoudre des problèmes très complexes (Zadeh, 1997).
- Certains préconisent que la logique classique conjointement avec la théorie de probabilité soient suffisantes pour offrir tout ce que la logique floue fournit comme outils de modélisation des problèmes complexes (Cheeseman, 1985).
- Certains contestent carrément l'aspect logique lui-même de la logique floue (Elkan, 1993).

L'idée de Zadeh consiste à utiliser le modèle de l'esprit humain pour formaliser le processus que les nouveaux produits industriels doivent adopter afin d'améliorer leurs MIQs. En effet, l'être humain dispose d'une très forte capacité pour appréhender la complexité et pour manier des notations vagues et imprécises. Cette compétence est due à l'habilité des humains à manipuler des informations imprécises et incertaines. Un exemple typique est de garer un véhicule dans un stationnement. Les humains sont capables de stationner facilement un véhicule car sa position et son orientation finales sont souvent spécifiées d'une manière imprécise plutôt qu'exacte; sinon, nous pourrions imaginer la difficulté d'un stationnement si on exigait un degré de précision très élevé.

Ainsi, Zadeh a initié le développement de la logique floue dont l'objectif principal est d'imiter les fonctionnalités de l'esprit humain (Zadeh, 1965). Il résume l'objectif de la logique floue par «The construction of smarter machines ». Aussi, Dubois et Prade, qui sont parmi les pionniers de la logique floue, affirment-ils (Dubois et Prade, 1991):

> «The main motivation of fuzzy set theory is apparently the desire to build up a formal, quantitative framework that captures the vagueness of human knowledge as it is expressed via natural language»

Durant la dernière décade du 20ème siècle, la logique floue s'est confirmée comme étant un outil adéquat pour le traitement des imprécisions et des incertitudes dans les systèmes intelligents. Le nombre de conférences internationales ainsi que le nombre de revues spécialisées en logique floue se compte actuellement par dizaines. Au niveau industriel, les différentes applications de la logique floue ont bien montré son utilité dans beaucoup de domaines tels que la robotique et le contrôle des automatismes de processus. Dans les sections suivantes de ce chapitre, nous présentons les concepts principaux de la logique floue que nous utiliserons dans le développement de notre modèle intelligent d'estimation des coûts.

## 2.4 ENSEMBLE FLOU

### 2.4.1 Utilité et définition

Selon Zadeh, la logique floue est la théorie des ensembles flous (Zadeh, 1994). La théorie des ensembles flous est une théorie mathématique dont l'objectif principal est la modélisation des notions vagues et incertaines du langage naturel. Ainsi, elle évite les inadéquations de la théorie des ensembles classiques quant au traitement de ce genre de connaissances.

La caractéristique fondamentale d'un ensemble classique est la frontière abrupte entre deux catégories d'éléments: ceux qui appartiennent à l'ensemble et ceux qui n'appartiennent pas à cet ensemble; ils appartiennent plutôt à son complémentaire. La relation d'appartenance est représentée dans ce cas par une fonction μ qui prend des valeurs de vérité dans la paire {0, 1}. Ainsi, la fonction d'appartenance d'un ensemble classique A est définie par:

$$m_A(x) = \begin{cases} 1 & si\ x \in A \\ 0 & si\ x \notin A \end{cases} \qquad \text{(Équation 2.1)}$$

Cela signifie qu'un élément $x$ est soit dans $A$ ($\mu_A(x) = 1$) ou non ($\mu_A(x) = 0$). Or dans plusieurs situations, il est parfois ambigu que $x$ appartienne ou non à $A$. Par exemple, considérons

l'ensemble *A* représentant les PCs qui sont *trop chers* pour une population d'étudiants. Après une enquête menée au sein de cette population, un PC ayant un prix supérieur ou égal à 2 500 $ sera déclaré *trop cher*, quand un prix inférieur ou égal à 1 000 $ n'est pas *trop cher*. Il existe un nombre important de PCs ayant un prix entre ces deux limites. Dans cet intervalle, on peut utiliser des valeurs, comprises strictement entre 0 et 1, pour classifier ces prix comme étant partiellement *trop cher*. Cette classification permettra de définir une nouvelle fonction d'appartenance, $\mu_A(x)$, associée à l'ensemble *A* représentant les PCs *trop chers*. $\mu_A(x)$ indique la valeur de vérité de la proposition *un PC est trop chers*. Si $\mu_A(x)$ est égal à 1 alors il est sûr et certain que *x* est dans *A;* $\mu_A(x)$ est égal à 0 implique que sûr et certain *x* n'appartient pas à *A;* $\mu_A(x)$ est strictement entre 0 et 1 implique que *x* appartient à *A* avec un degré de vérité égal à $\mu_A(x)$. *A* est donc l'ensemble flou associé à la valeur linguistique *trop cher*. *A* sera noté par:

$$A = \int_X m_A(x)/x$$

où $\mu_A(x)$ est la fonction d'appartenance à *A* et *X* est l'ensemble de toutes les valeurs possibles de *x*: l'univers de discours de *x* (l'ensemble des réels IR dans la plupart des cas). La figure 2.1 illustre la différence entre la représentation de la valeur *trop cher* par un ensemble classique et par un ensemble flou.

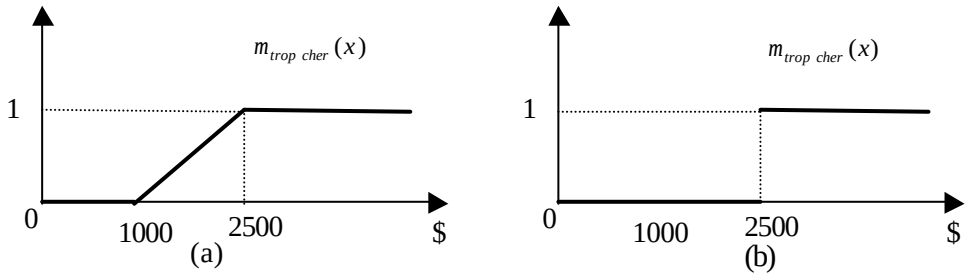



**Figure 2.1** Ensemble flou vs ensemble classique pour la valeur linguistique *trop cher* (Jager, 1995)

En général, un ensemble flou est utilisé pour modéliser l'incertitude et les imprécisions dans la connaissance:

1- Incertitude: la fonction d'appartenance $\mu_A(x)$ est utilisée pour indiquer le degré de vérité de la proposition *x est A.* Dans ce cas, on connaît la valeur de x mais on ne connaît pas à quel ensemble elle appartient; en effet x peut appartenir à plusieurs ensembles avec différents (ou mêmes) degrés d'appartenance. L'ensemble flou modélise alors ici l'aspect *incertain* de la connaissance.

2- Imprécision: la fonction d'appartenance $\mu_A(x)$ est une distribution de possibilité dans l'espace de toutes les valeurs possibles de *x*. Dans ce cas, on connaît l'ensemble (ou les ensembles) auquel appartient x mais on ne connaît pas la valeur exacte de *x*; $\mu_A(x')$ représente la possibilité pour que $x=x'$. L'ensemble flou modélise alors ici l'aspect *imprécis* de la connaissance.

2.4.2 Opérations sur les ensembles flous

Comme dans le cas des ensembles classiques, plusieurs opérations sont possibles sur les ensembles flous telles que l'intersection, l'union et le complément. On retrouve donc dans la littérature une multitude de définitions des opérateurs implémentant ces opérations (Tab. 2.2). Par exemple, l'opération d'intersection est implémentée par les opérateurs du type *T-norms* (*Triangular norms*); l'opération d'union est implémentée par des opérateurs du type *T-conorms* (*Triangular conorms*). Une fonction de $[0,1]\times[0,1]$ vers $[0,1]$ est dite *T-norm* si elle satisfait les quatre critères suivants (Jager, 1995):

T-1: $T(a, 1) = a$

T-2: $T(a, b) \leq T(c,d)$ si $a \leq c$ , $b \leq d$

T-3: $T(a,b) = T(b,a)$

T-4: $T(T(a,b),c) = T(a,T(b,c))$

Un exemple d'une fonction *T-norm* est la fonction *f(a,b)=min(a,b;* Figure 2.2a). D'ailleurs, toute fonction *f* du type *T-norm* satisfait la condition suivante:

$$f(a,b) \leq min(a,b)$$

Une fonction de [0,1]×[0,1] vers [0,1] est dite *T-conorm* (ou *S-norm*) si, en plus des trois critères T-2, T-3 et T-4, elle satisfait la condition suivante:

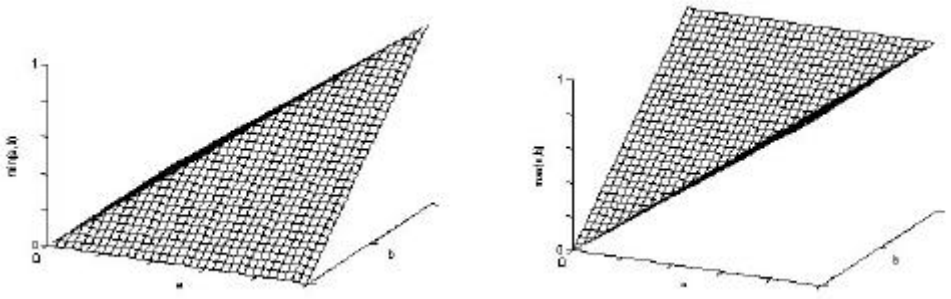
S-1: S(a,0) =a

Un exemple d'une fonction *T-conorm* est la fonction *f(a,b)=max(a,b)* (Fig. 2.2b). D'ailleurs, toute fonction *f* du type *T-conorm* satisfait la condition suivante:

$$max(a,b) \leq f(a,b)$$

Un opérateur qui définit le complément d'un ensemble flou A, C(A), doit vérifier les conditions suivantes:

C-1 C(0) = 1
C-2 C(a) < C(b) quand a>b
C-3 C(C(a)) = a



(a) (b)

**Figure 2.2** (a) Opérateur *min* en tant que borne supérieure de tous les opérateurs *T-norm.* (b) Opérateur *max* en tant que borne inférieure de tous les opérateurs *T-conorm.*

**Tableau 2.2**
Examples des opérateurs *T-norms*, *S-norms* et le complément

| Références | T(a,b) | S(a,b) | C(a) |
|---|---|---|---|
| Zadeh, 1973 | min (a,b) | max (a,b) | 1-a |
| Lukasiewicz, 1976 | max (a+b-1,0) | min (a+b-1,0) | Sugeno, 1977 |
| Bandler et Kohout, 1980 | Ab | a+b-ab | $\frac{1-a}{1+\lambda a}, \quad \lambda > 0$ |
| Hamacher, 1978 | ab/(γ+(1-γ); a+b-ab) , γ≥0 . | (a+b –(2-γ)ab)/(1-(1-γ)ab) , γ≥0 . | Yager, 1980 |
| Weber, 1983 | a si b=1<br>b si a=1<br>0 sinon | a si b=0<br>b si a=0<br>1 sinon | $\sqrt[P]{(1-a^P)}, P > 0$ |

2.4.3 Propriétés des ensembles flous

Dans cette section, nous présentons quelques propriétés des ensembles flous que nous utiliserons dans les chapitres 4 et 5. *A* désigne un ensemble flou, *X* est son univers de discours et $m$ est sa fonction d'appartenance:

- Hauteur d'un ensemble flou: La hauteur de *A*, *hgt(A)*, est définie par :

$$hgt(A) = \sup_{x \in X} m_A(x) \qquad \text{(Équation 2.2)}$$

*hgt(A)* représente le degré le plus élevé d'appartenance à A . Un ensemble flou A est dit *normal* si *hgt(A)=1*.

- Noyau d'un ensemble flou: Le noyau de *A*, *core(A)*, est défini par:

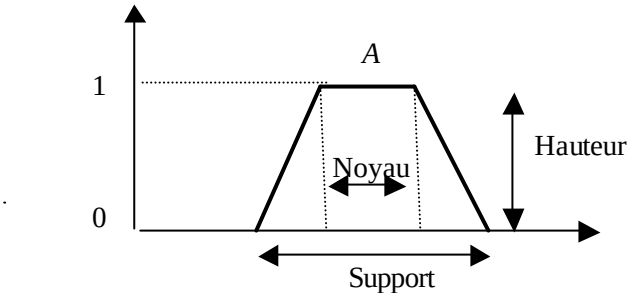
$$Core(A) = \{x \in X / m_A(x) = 1\} \qquad \text{(Équation 2.3)}$$

Le *core(A* ) contient tous les éléments qui appartiennent sûr et certain à *A* (leurs degrés d'appartenance sont égaux à 1) .

- Support d'un ensemble flou : Le support de *A*, *supp(A)*, est défini par :

$$Supp(A) = \{x \in X / m_A(x) > 0\} \qquad \text{(Équation 2.4)}$$

Le *supp(A )* contient tous les éléments qui appartiennent à *A* avec des degrés strictement supérieurs à 0 (appartenance partielle ou complète).




**Figue 2.3** *Hauteur*, *Noyau* et *support* d'un ensemble flou.

- Le α-cut d'un ensemble flou ( le niveau α de *A*): Le α-cut de *A*, *a-cut(A)*, est défini par :

$$a - cut(A) = \{x \in X / m_A(x) \geq a\} \qquad \text{(Équation 2.5)}$$

Le *a-cut(A)* contient tous les éléments qui appartiennent à *A* avec des degrés d'appartenance supérieurs ou égaux à $a$. De la même façon, le *a-cut-strict(A*) est défini en remplaçant $m_A(x) \geq a$ par $m_A(x) > a$ dans l'équation 2.5. Ainsi, on peut noter que:

$$Core(A) = 1 - cut(A)$$

$$Supp(A) = 0 - cut - strict(A)$$

- Ensemble flou convexe : *A* est dit convexe si :

$$\forall x_1, x_2, x_3 \in X, x_1 \leq x_2 \leq x_3 \quad \text{alors} \quad m_A(x_2) \geq \min(m_A(x_1), m_A(x_3)) \qquad \text{(Équation 2.6)}$$

- Partition floue: Soient N ensembles flous $A_j$ du référentiel $X$. ($A_1$, $A_2$,.., $A_j$,., $A_N$) est dite une *partition floue* si:

$$\forall x \in X \quad \sum_{j=1}^{N} m_{A_j}(x) = 1 \quad avec \ A_j \neq f \quad et \quad A_j \neq X \quad \forall \quad 1 \leq j \leq N \qquad \text{(Équation 2.7)}$$

La figure 2.5 illustre un exemple d'une partition floue formée de cinq ensembles flous.

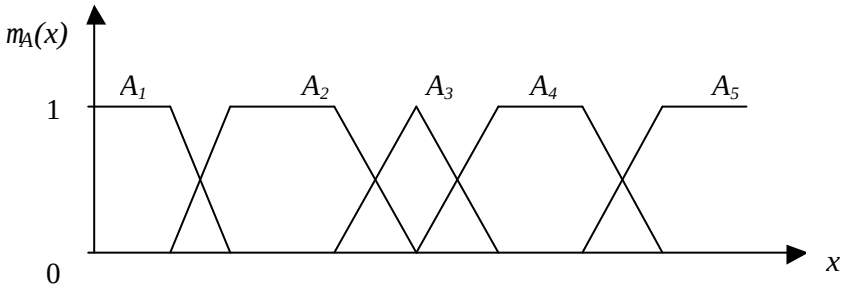



**Figure 2.4** Exemple d'une partition floue formée de cinq ensembles flous.

## 2.5 VARIABLES LINGUISTIQUES ET VALEURS NUMÉRIQUES

Une variable linguistique est une variable dont les valeurs associées sont linguistiques plutôt que numériques (Zadeh, 1997). Par exemple, la variable linguistique *âge* peut être évaluée par les trois valeurs linguistiques suivantes: *jeune*, *moyen* et *vieux*. La notion de variable linguistique suppose donc l'existence d'un univers de discours $X$ et d'un ensemble de valeurs linguistiques D associé à la variable linguistique étudiée. Pour la variable linguistique *âge*, $X$ est l'ensemble des nombres réels positifs $IR^+$ et D = {*jeune*, *moyen*, *vieux*}. En logique floue, les valeurs linguistiques sont représentées par des ensembles flous plutôt que par des ensembles classiques (intervalles ou nombres). L'avantage principal de cette représentation est qu'elle ressemble, a priori, à celle utilisée par les humains quand ils interprètent les valeurs linguistiques. Considérons, par exemple, le cas de la valeur linguistique *jeune*; elle peut être représentée de trois façons différentes (Fig. 2.5).

La représentation de la valeur linguistique *jeune* par une seule valeur numérique (27 dans cet exemple) considère que seules, les personnes ayant un âge exactement égal à 27, appartiennent à l'ensemble *jeune* (Fig. 2.5c). Or, ceci n'est pas conforme à la vision de la plupart d'entre nous. En effet, on considèrera souvent les personnes ayant un âge approximativement égal à 27 comme des jeunes. Cela nous oblige donc à définir une marge de tolérance autour de la valeur 27. La représentation de la valeur linguistique, *jeune*, par un intervalle remédie à cette limitation (Fig. 2.5b). Cependant, cette représentation montre le même problème que celle utilisant une seule valeur numérique aux limites de l'intervalle. En effet, la transition aux limites de l'intervalle est brusque. Or, ceci n'est pas conforme à notre intuition car nous imaginons souvent que la transition doit être graduelle plutôt que brusque. La représentation par un ensemble flou évite ce problème et permet la tolérance des imprécisions aux limites de l'intervalle (Fig. 2.5a).

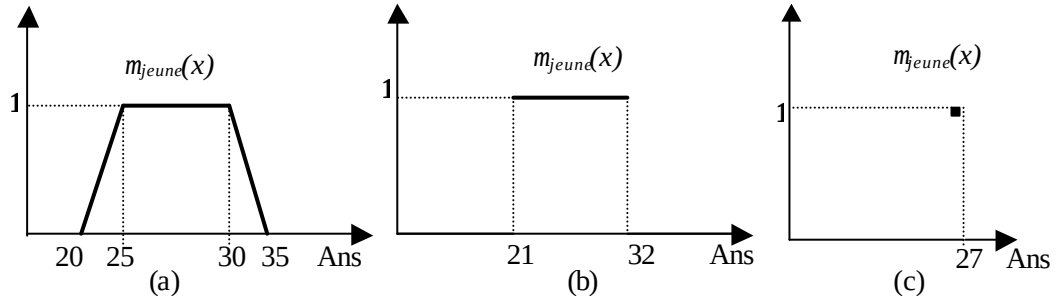



**Figure 2.5** (a) Représentation par un ensemble flou, (b) Représentation par un intervalle classique, (c) Représentation par un nombre.

## 2.6 LOGIQUE FLOUE, THÉORIE DE PROBABILITÉ ET THÉORIE DE MESURE

Bien que plusieurs débutants confondent souvent la fonction d'appartenance de l'ensemble flou avec une loi de probabilité parce qu'elles affectent toutes les deux des valeurs dans l'intervalle [0,1]. Il s'agit en réalité de deux disciplines totalement différentes. Un degré d'appartenance constitue une mesure d'incertitude ou de croyance par rapport à une notion vague telle que *grand*, *intelligent* et *cultivé*. Cette mesure indique que selon l'univers du discours considéré et selon les conventions adoptées, les individus sont *grands*, *intelligents* et *cultivés*. Par contre, une probabilité est inévitablement associée à la notion d'événement (ou valeurs subjectives attribuées à l'événement). C'est un poids associé à l'occurrence d'un

événement (ou jugement subjectif sur l'événement); il y a donc incertitude et attente de voir celui-ci se réaliser. Les chances de réalisation de cet événement généralement précis dépend de la valeur attribuée à la probabilité correspondante. L'événement peut être parfois flou; un exemple en est la probabilité associée à la proposition *le prochain président sera jeune.* Si par exemple la réponse est 0,5, cela implique que le prochain président aura un degré non nul d'appartenance à l'ensemble flou *jeune* avec une probabilité de 0,5; son âge peut donc être 27, 30 ou 33. Bref, n'importe quelle valeur ayant un degré d'appartenance, à l'ensemble flou *jeune*, différent de 0.

Une autre confusion peut également s'établir entre les valeurs d'une fonction d'appartenance et les valeurs d'une mesure. Une mesure génère des valeurs réelles suite à l'application d'un ensemble de règles, d'étapes et/ou de formules qui constituent le processus de mesurage en question (voir Section 3.2). Par exemple, pour mesurer le poids d'une personne, il faut tout d'abord ajuster la balance, monter et se tenir tout droit sur la balance, l'opérateur se mettant en face en lisant la valeur pointée par la flèche rouge. Certaines mesures génèrent des valeurs dans l'intervalle [0,1]; ce qui laisse croire aux débutants que c'est la même chose que les valeurs d'une fonction d'appartenance. Bien que les deux valeurs peuvent être dans l'intervalle [0,1], leur signification est totalement différente.

L'exemple suivant illustre la différence entre la valeur d'une fonction d'appartenance, la valeur d'une loi de probabilité et la valeur d'une mesure. Nous le présentons ici dans sa version originale afin d'éviter toute mauvaise traduction[3]:

> «Recently, he following "popular" explaination of the differences between probability, measure theory, and fuzzy sets is proposed:
> *Question*: If there a salami sandwich in the refrigerator?
> *Answer*: 0.5
> If "probability", then there either is or isn't, with probability one half.
> If "measure", then there is half a salami sandwich there.
> If "fuzzy", then there is something there, but it isn't really a salami sandwich. Perhaps it is some other kind of sandwich or salami without the bread and so one.»

[3] Cet exemple est le plus utilisé au sein de la communauté de la logique floue, à travers son *mailing list* BISC (*Berckely Initiative Soft Computing*): bisc-group@cs.berkeley.edu, pour expliquer la différence entre une fonction d'appartenance d'une part, et une loi de probabilité ou une mesure d'autre part.

## 2.7 LA FACETTE LOGIQUE DE LA LOGIQUE FLOUE

Zadeh distingue quatre facettes de la logique floue (Zadeh, 1997):

- la facette logique,
- la facette de la théorie des ensembles,
- la facette relationnelle, et
- la facette épistémologique.

Dans cette section, nous présentons la facette logique en comparaison avec d'autres systèmes logiques existant avant l'apparition de la logique floue.

La logique classique (logique propositionnelle, logique des prédicats, algèbre de Boole) est caractérisée par ses deux valeurs de vérité (vraie et fausse) et ses règles d'inférence telles que le *modus ponens* et le *modus tollens*. Cette logique est inappropriée pour une description fine et souple des connaissances vagues, imprécises et incertaines. D'autres systèmes logiques tels que la logique temporelle introduisant le paramètre *temps* dans les valeurs de vérité, et la logique modale, introduisant le concept de modalité, ne constituent pas réellement une véritable alternative de la logique classique dans les situations où les connaissances sont imprécises et incertaines (Tong-Tong, 1995). En effet, elles adoptent toujours les deux valeurs de vérité (vraie et fausse). La vraie révolution était celle de la logique multivalente et de la logique floue. En effet, ces deux systèmes ne respectent particulièrement plus le principe du tiers-exclu et sont caractérisés par un nombre de valeurs de vérité supérieur à deux et par une généralisation de l'implication et des règles classiques d'inférence dont notamment le *modus ponens* et le *modus tollens*.

La logique multivalente ou la logique n-aire est une généralisation de la logique binaire. En effet, ses valeurs de vérité dont le nombre n est supérieur à deux, sont des éléments de l'intervalle [0,1]. Ainsi, on parle de la logique trivalente (ternaire) lorsque n=3, ou de logique endécadaire pour n=11. Contrairement à la logique classique, il existe différentes logiques multivalentes proposées par divers auteurs. Leurs différences viennent essentiellement de la définition de l'implication dans chacun des systèmes. Ainsi, il existe plusieurs logiques ternaires telles que celle introduite par Lukasiewicz, Gödel, Goguen, Lee, Kleene et autres.

La logique multivalente est apparue pour répondre aux préoccupations de nombreux philosophes, mathématiciens et logiciens. Son objectif était de dépasser le principe du tiers-exclu. Par exemple, pour Lukasiewicz la motivation essentielle d'extension de la logique binaire, en introduisant une troisième valeur de vérité, était de répondre au futur contingent. En effet, appliquer le vrai et le faux aux événements futurs suppose qu'ils sont déjà déterminés et exclut de ce fait leur contingence. Selon Lukasiewicz la proposition *Je passerai Noël à Varsovie* ne peut être ni vraie ni fausse au présent, car elle exprimerait dans le premier cas un futur nécessaire et dans le deuxième cas un futur impossible. Mais si l'on peut considérer que la nécessité de la généralisation de la logique classique en une logique plus riche a été la préoccupation commune des différents chercheurs, on constate que les motivations et les interprétations de la troisième valeur de vérité introduite dans la logique ternaire, par exemple, ne sont pas les mêmes. En effet, pour Lukasiewicz cette troisième valeur représente une valeur intermédiaire, pour Bochvar c'est l'absurde (le non-sens) et pour Kleene, l'indécision.

Un exemple de la logique n-aires proposé par Gödel, Goguen et d'autres consiste à introduire n états ou valeurs de vérité de l'intervalle [0,1]. Ces valeurs de vérité sont généralement définies par $T_n$ = {0, 1/n-1, 2/n-1,…, n-2/n-1, 1}. Ainsi, $T_2$={0,1} représente la logique binaire, $T_3$={0,1/2,1} représente la logique ternaire et ainsi de suite. Certes la logique multivalente représente une extension importante de la logique classique mais son champ d'application reste tout de même limité. En effet, la logique multivalente peut être schématiquement caractérisée par l'introduction d'une ou de plusieurs valeurs de vérité supplémentaires et par une extension des connecteurs (intersection, union, complémentation, etc.). Cette généralisation est insuffisante lorsqu'on aborde certains domaines d'application où les connaissances manipulées sont vagues, imprécises et incertaines. Aussi, une valeur de vérité ajoutée par une logique multivalente n'est qu'un nombre relationnel entre 0 et 1 représentant un degré de vérité d'une proposition de la logique n-aires considérée; la dénomination d'une valeur intermédiaire entre 0 (faux) et 1 (vrai) n'est pas unique; elle peut être *indéterminé, indécidable, ni vrai ni faux* ou *absurde* selon la logique considérée.

La logique floue est aussi une généralisation de la logique binaire. Elle se distingue de la logique multivalente par (Zadeh, 1997; Tong-Tong, 1995):

- Elle est liée à la théorie des ensembles flous contrairement à la logique multivalente qui reste toujours liée à la théorie des ensembles classiques.

- En logique n-aires, une valeur de vérité est généralement représentée par un nombre relationnel k/n-1; par contre dans la logique floue, la valeur de vérité d'une proposition floue (voir plus loin la définition d'une proposition floue) est un nombre réel de l'intervalle [0,1] qui représente le degré d'appartenance (ou de satisfaction) à une notion vague telles que *jeune*, *grand* et *beau* énoncée dans la proposition floue.

- La logique floue offre, en plus d'une représentation convenable des connaissances imprécises et incertaines, un ensemble de mécanismes d'inférence appropriés à ce genre de représentation (section suivante).

## 2.8 RAISONNEMENT FLOU (RAISONNEMENT APPROXIMATIF)

Le raisonnement flou ou approximatif constitue une des composantes importantes de la logique floue. L'idée de Zadeh est de simuler le raisonnement approximatif chez les humains. Il formula en 1979, le raisonnement flou comme suit (Zadeh, 1979):

> «The theory of approximate reasoning is concerned with the deduction of possibly imprecise conclusions from a set of imprecise premises»

L'esprit humain présente un modèle de raisonnement très puissant puisqu'il est capable de représenter et de raisonner à partir de connaissances précises ou floues. Selon Zadeh, cette compétence est due à l'habileté des humains à manipuler des perceptions en plus des mesures. La différence entre une perception et une mesure est qu'en général une mesure est *crisp* (précise) tandis qu'une perception est *fuzzy* (floue). Nos perceptions sont souvent décrites au moyen de ce puissant outil qu'est le langage naturel. Tout le monde est convaincu du rôle du langage naturel dans l'intelligence. C'est grâce à cet outil que nous arrivons à résoudre et à manipuler des situations très complexes dans notre vie quotidienne (conduire une voiture, donner un cours, vendre une marchandise, etc.). Notre langage naturel présente

des caractéristiques très intéressantes qui représentent un atout pour notre intelligence. Il supporte l'ambiguïté, les imprécisions, l'incertain et la partialité dans la description de nos connaissances ainsi que dans notre raisonnement. Cependant, la plupart des autres langages formels utilisés par l'IA symboliste ne peuvent traiter ou supporter ces qualifications. Par conséquent, peu importe la capacité d'abstraction d'un langage formel, on peut remarquer que dans plusieurs cas, le passage de la connaissance, telle qu'elle est décrite dans la réalité, à sa représentation dans le langage formel, est restrictif , c'est-à-dire qu'on peut observer une grande différence entre ce qui est réellement perçu (ce qui relève de la réalité) et ce que le langage formel nous communique de cette réalité.

Dans ses travaux récents, Zadeh a initié le développement d'une nouvelle méthodologie de *computation* par des mots (*Computing with words* - CW) dont l'objectif est d'imiter le langage naturel utilisé par les humains. Il s'agit simplement, pour le moment, d'un sous ensemble du langage naturel: le langage des propositions. La différence entre l'approche de Zadeh (CW) et les langages formels de l'IA symboliste est que: (1) le CW est basé sur la logique floue alors que les langages de l'IA symboliste sont basés sur la logique classique; (2) le symbole dans l'approche de Zadeh est le *mot* alors que dans les langages de l'IA, il peut être autre chose (un nombre par exemple).

Le CW fournit un modèle mathématique pour traiter le langage naturel dont le *mot* constitue l'élément principal. Ce que les brillants cerveaux de l'IA symboliste n'ont pas remarqué c'est que les humains effectuent chaque jour des tâches très complexes en n'utilisant seulement le langage naturel comme outils de communication. C'est à l'aide de ce langage qu'on décrit la plupart de nos perceptions telles que la perception de la vitesse, de la taille et de la forme d'un objet. Le CW fournit une méthodologie pour ce que Zadeh nomme le *Computationnel Theory of Perceptions* (CTP). En effet, dans le CTP, les perceptions sont exprimées par des propositions du langage naturel. Ainsi, ces perceptions sont traitées par des méthodes basées sur le CW pour répondre à une question exprimée elle aussi en langage naturel. Le CW fournit donc les outils de base pour évaluer et raisonner selon des approches approximatives plutôt qu'exactes. Zadeh illustre ceci par deux exemples (Zadeh, 1996; Zadeh, 2001):

1- Considérons les deux propositions suivantes: *Most young men are healthy* et *Robert is a young men*. Que peut-on dire de la santé de Robert? Ni la logique de prédicats ni la théorie de probabilité ne sont capables de répondre à cette question.

2- Considérons un sac contenant des balles de tailles différentes: la plupart des balles sont de grande taille et peu d'entre elles sont de petite taille. Quelle est la probabilité en tirant du sac, au hasard, une balle de taille ni grande ni petite? Dans ce cas, pour répondre à cette question, nous devons faire appel à la *computation* par des perceptions pour définir la signification de *La plupart des balles sont de grande taille, peu d'entre elles sont de petite taille* et *ni grande ni petite*.

En CW, la connaissance est représentée par des propositions en langage naturel. Ces propositions sont les descriptions linguistiques de nos perceptions. Cette collection de connaissances constitue l'ensemble initial de données (*Initial Data Set* -IDS-). À partir de cet ensemble IDS, on souhaite répondre à une question formulée elle aussi en langage naturel. Notre réponse doit être aussi exprimée en langage naturel et sera notée par *Terminal Data Set* (TDS). Donc, le rôle du CW est la transformation du IDS pour obtenir le TDS (Fig. 2.6). Par exemple, à partir de l'IDS suivant: *Most Swedes are tall*, que peut-on dire sur la taille moyenne des Suédois?

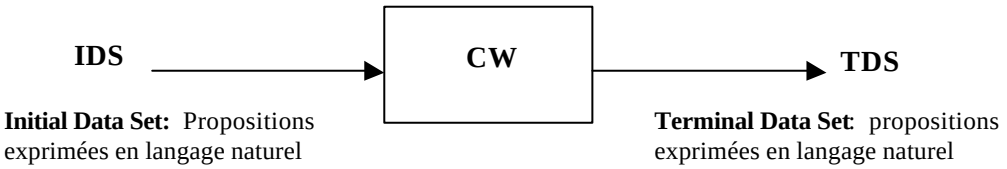



**Figure 2.6** Rôle du CW.

La figure 2.7 explique la structure conceptuelle du CW. Le point de départ du CW consiste à représenter la signification d'une proposition formulée en langage naturel par une forme canonique qui montre la variable linguistique ainsi que sa contrainte floue. La forme canonique d'une proposition met en alors évidence d'une manière explicite, la contrainte qui se trouve implicitement dans la proposition[4]. Ainsi, on représente toutes les propositions de

[4] Par exemple, la forme canonique de la proposition *Marie est jeune* est *Âge(Marie) est jeune*. Ici la variable linguistique est la variable *âge*, la contrainte floue est *jeune.*

l'IDS sous leurs formes canoniques. Après, on utilise les règles d'inférence floues pour déduire des contraintes floues décrivant le TDS. Ces contraintes floues du TSD seront reformulées en langage naturel en utilisant des approximations linguistiques.

Le processus de raisonnement du CW est basé sur les règles d'inférence floues. On retrouve donc tous les types de règles d'inférence telles que le *modus ponens* flou, la composition, la projection, le principe d'extension, la conjonction, la disjonction, etc. Ces mécanismes d'inférence manipulent des règles floues. Nous présentons, par la suite, quelques éléments du CW qui serviront pour une bonne compréhension des chapitres suivants.

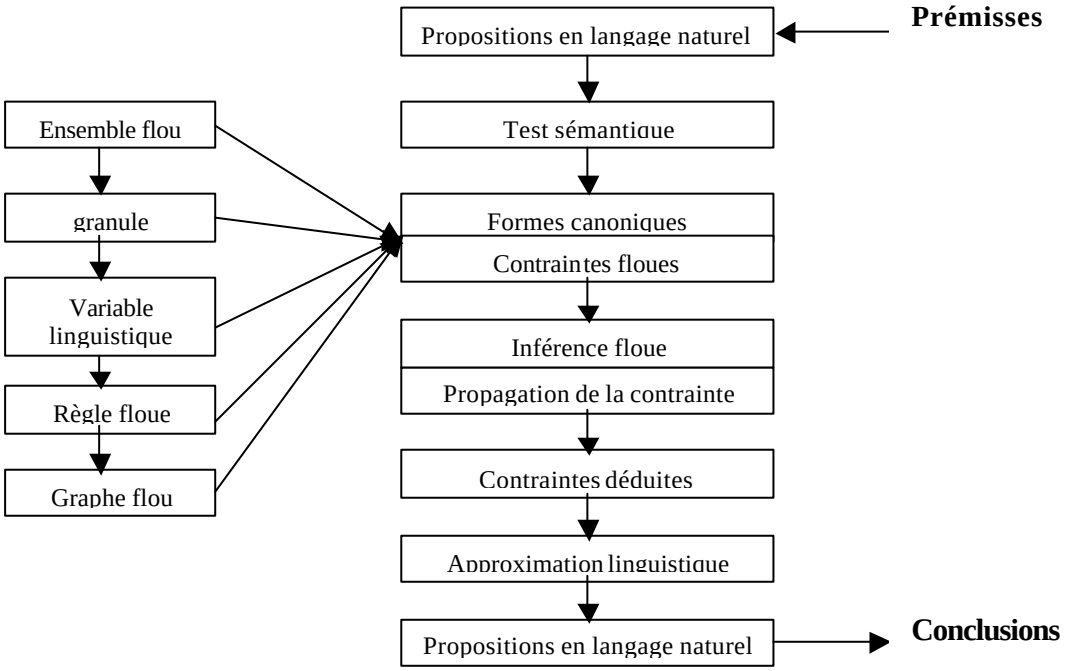



**Figure 2.7** La structure conceptuelle du CW (Zadeh, 2001).

2.8.1 Proposition floue

Une proposition floue est un énoncé contenant un ou plusieurs prédicats flous selon qu'elle est simple ou composée. Sa valeur de vérité appartient à l'intervalle [0,1] contrairement à la logique classique où la valeur de vérité d'une proposition est soit vraie (1) ou fausse (0). Les

propositions floues sont à la base du raisonnement flou. Elles sont similaires à celles qu'on utilise dans notre vie quotidienne. En effet, elles sont exprimées en langage naturel. Ainsi, elles permettent au raisonnement flou d'être assez proche du raisonnement approximatif adopté par les humains. Des exemples de propositions floues sont:

*La charge est lourde*
*Il fait chaud dans la salle de réunions*
*Il est jeune*
*Il fait beau aujourd'hui*
*Le marié est grand*

Les propositions floues peuvent être combinées, comme dans le cas de la logique classique, par des connecteurs logiques tels que le *et* et le *ou*. Ces deux connecteurs logiques sont implémentés respectivement par des opérateurs de type *T-norms* et *T-conorms* (Tab. 2.2). Dans la littérature, il n'y a pas de règles claires à suivre pour le choix de l'opérateur *T-norm* (ou *T-conorm*) implémentant l'intersection (ou l'union). Souvent, le choix de l'opérateur dépend de la signification et du contexte des propositions ainsi que des relations existantes entre elles. Cependant, les opérateurs de Zadeh, ceux de Lukasiewicz ainsi que ceux de Bandler et Kohout sont les plus utilisés dans la logique floue (Jager, 1995).

Une proposition floue composée est constituée de propositions floues simples reliées entre elles par des connecteurs logiques tels que *et* et *ou*. Par exemple, *Ali est jeune et Ali est grand* est une proposition qui contient deux propositions floues simples: *Ali est jeune* et *Ali est grand*. Une autre manière de combiner des propositions floues simples consiste à utiliser l'implication. Une proposition floue utilisant l'implication est de la forme: *Si x est A alors y est B.* Par exemple, *Si Ali est jeune alors il est fort*. Il existe, comme on le verra dans la section suivante, différents types d'interprétation d'une implication floue. Une proposition floue du type Si-Alors est appelée une *règle floue*.

2.8.2 Règle floue

Une règle floue est une affirmation (Si–Alors) dont la prémisse et la conséquence sont des propositions floues ou des combinaisons de propositions floues par des connecteurs logiques (souvent le *et* et le *ou*). Par exemple, la règle floue *si* $x_1$ *est* $A_1$ *et* $x_2$ *est* $A_2$ *alors y est B* est formée d'une prémisse composée de deux propositions floues (*si* $x_1$ *est* $A_1$ *et* $x_2$ *est* $A_2$) combinées par le connecteur logique *et*, et une conséquence formée par une proposition floue simple (*y est B*). Des exemples de règles floues sont:

*S'il fait très chaud alors ouvrir la fenêtre.*
*Si la chaussée est mouillée alors ralentir.*
*Si la maison est neuve et si elle n'est pas loin de la mer alors son coût est très élevé.*

Une règle floue (Si-Alors) est représentée par une implication floue ayant la même fonctionnalité que celle utilisée dans la logique classique. Par exemple, la règle floue:

*si* $x_1$ *est* $A_1$ *et* $x_2$ *est* $A_2$ *alors y est B*

où $A_1$, $A_2$, et $B$ sont des ensembles flous représentés respectivement par les fonctions d'appartenances $m_{A_1}(x_1), m_{A_2}(x_2) et m_B(y)$, peut être représentée par la relation suivante:

$$R = I(T(A_1, A_2), B) \qquad \text{(Équation 2.8)}$$

où $T(A_1, A_2)$ est la conjonction *et* implémentée par un opérateur *T-norm* et I est une fonction qui représente l'implication floue de la règle Si–Alors. R peut donc être représentée par la fonction d'appartenance:

$$m_R(x_1, x_2, y) = I(T(m_{A_1}(x_1), m_{A_2}(x_2)), m_B(y)) \qquad \text{(Équation 2.9)}$$

### 2.8.3 Implication floue

Dans la littérature, on retrouve plusieurs représentations d'une implication floue A→B (Dubois et Prade, 1991). Cependant, selon Jager, elles peuvent être regroupées en deux catégories différentes (Jager, 1995):

- Les implications floues interprétées par une conjonction, c'est-à-dire $A \rightarrow B \equiv A \cap B$. Par conséquent, la fonction I représentant l'implication floue serait de la forme *I(a,b)= T(a,b)* où T est un opérateur *T-norm*. Les figures 2.8c et 2.8d montrent deux exemples d'implications implantées respectivement par l'opérateur *min* (implication de Mamdani) et l'opérateur *produit* (implication de Larsen).

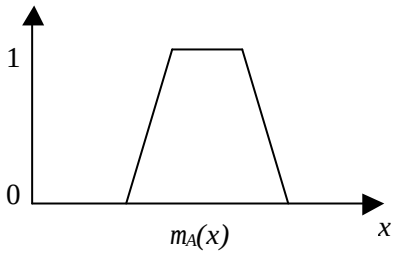
- Les implications floues interprétées par une implication classique, c'est-à-dire $a \rightarrow b \equiv \neg a \vee b$. Par conséquent, la fonction I représentant l'implication floue serait de la forme $I(a,b) = S(C(a),b)$ où S est un opérateur *S-norm* (*T-conorm*) et *C* est le complément flou. Les figures 2.8e et 2.8f montrent deux exemples d'implications implantées respectivement par l'opérateur *max* (implication de Kleene-Dienes) et l'opérateur *min* (implication de Goguen).

En combinant les deux interprétations d'une implication floue citées ci-dessus, d'autres types d'interprétations ont été proposés:

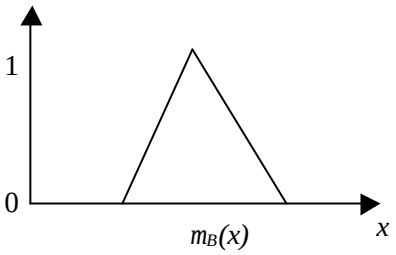
- Les implications floues interprétées comme suit: $A \rightarrow B \equiv \neg A \vee (A \cap B)$. Ce type d'interprétations est souvent référé par *QL-implication* (*Quantum Logic implication*). Par conséquent, la fonction I représentant l'implication floue serait de la forme $I(a,b) = S(C(a),T(a,b))$ où T est un opérateur *T-norm*, S est un opérateur *S-norm* et *C* est le complément flou.

- Les implications floues interprétées selon la règle du *modus tollens*, c'est-à-dire $A \rightarrow B \equiv \neg B \rightarrow \neg A$.

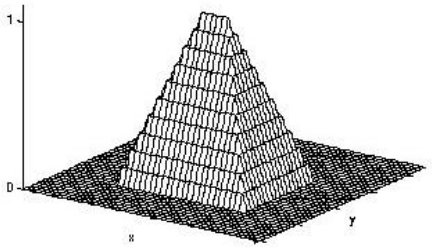
- Les implications floues interprétées selon les deux règles du *modus ponens* et du *modus tollens*, c'est-à-dire $A \rightarrow B \equiv (A \rightarrow B) \cap (\neg B \rightarrow \neg A)$.




(a)

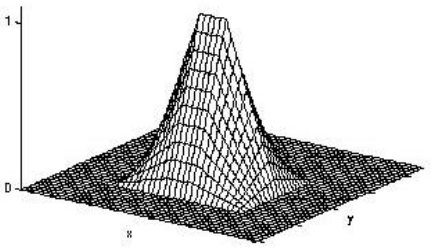



(b)



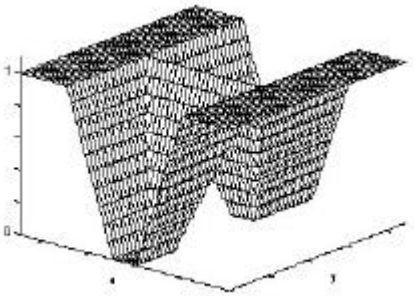
(c)

*I(a,b)=min (a,b)*



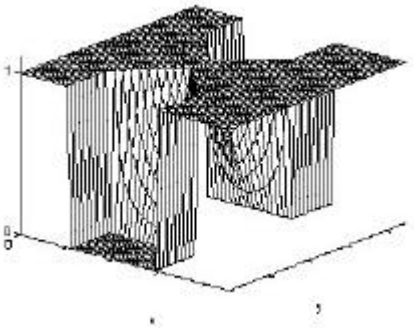
(d)

*I(a,b)=ab*



(e)

*I(a,b)=max(1-a,b)*



(f)

*I(a,b)=min(a/b,1)*

**Figure 2.8** Exemples de relations floues R= I(a,b) représentant la règle floue A→B, A et B sont deux ensembles flous (a) et (b). (c) Implication de Mamdani. (d) Implication de Larsen. (e) Implication de Kleene-Dienes. (f) Implication de Goguen.

### 2.8.4 Agrégation des règles floues

Dans les deux sections précédentes, nous avons expliqué le processus de présentation d'une règle floue (Si-Alors) par une relation floue. Cependant, dans les applications de la logique floue, plusieurs règles floues, plutôt qu'une seule, sont souvent utilisées pour reformuler la problématique étudiée. Le processus de combinaison d'un ensemble de règles floues, $r_k$, pour en faire une seule représentée par la relation R, s'appelle l'*agrégation*. Considérons $N_r$ règles floues, chacune à $N_x$ variables dans sa prémisse:

$r_1$: *si* $x_1$ *est* $A_{11}$ *et* …..…..*et* $x_{Nx}$ *est* $A_{Nx1}$ *alors y est* $B_1$
. .
$r_k$: *si* $x_1$ *est* $A_{1k}$ *et* ……..…..*et* $x_{Nx}$ *est* $A_{Nxk}$ *alors y est* $B_k$
. .
$r_{Nr}$: *si* $x_1$ *est* $A_{1Nr}$ *et* ……..…..*et* $x_{Nx}$ *est* $A_{NxNr}$ *alors y est* $B_{Nr}$

L'agrégation d'un ensemble de règles floues en une seule relation floue est effectuée en deux étapes. Premièrement, chaque règle floue $r_k$ est transformée en une relation floue $R_k$. Deuxièmement, les relations floues $R_k$ sont combinées pour obtenir une seule relation floue $R$. L'agrégation des règles $r_k$ dépend de l'interprétation adoptée pour les implications floues contenues dans ces règle $r_k$:

- Si l'implication floue est interprétée par une conjonction, l'opérateur d'agrégation est alors la disjonction: $R = \cup R_k$. Dans ce cas, l'opération de disjonction des relations floues $R_k$ peut être implémentée par n'importe quel opérateur *S-norm* (*S-norm aggregation*). La colonne de droite de la figure 2.9 montre un exemple d'agrégation de deux règles $R_1$ et $R_2$ en utilisant l'implication de Mamdani (opérateur *min*) et l'opérateur *max* pour l'agrégation (agrégation *max-min*).

- Si l'implication floue est interprétée par une implication classique alors l'opérateur de l'agrégation est la conjonction: $R = \cap R_k$. Dans ce cas, l'opération de conjonction des relations floues $R_k$ peut être implémentée par n'importe quel opérateur *T-norm* (*T-norm aggregation*). La colonne de gauche de la figure 2.9 montre un exemple d'agrégation de deux règles $R_1$ et $R_2$ en utilisant l'implication de Kleene-Dienes et l'opérateur *min* pour l'agrégation (agrégation min-Kleene-Dienes).

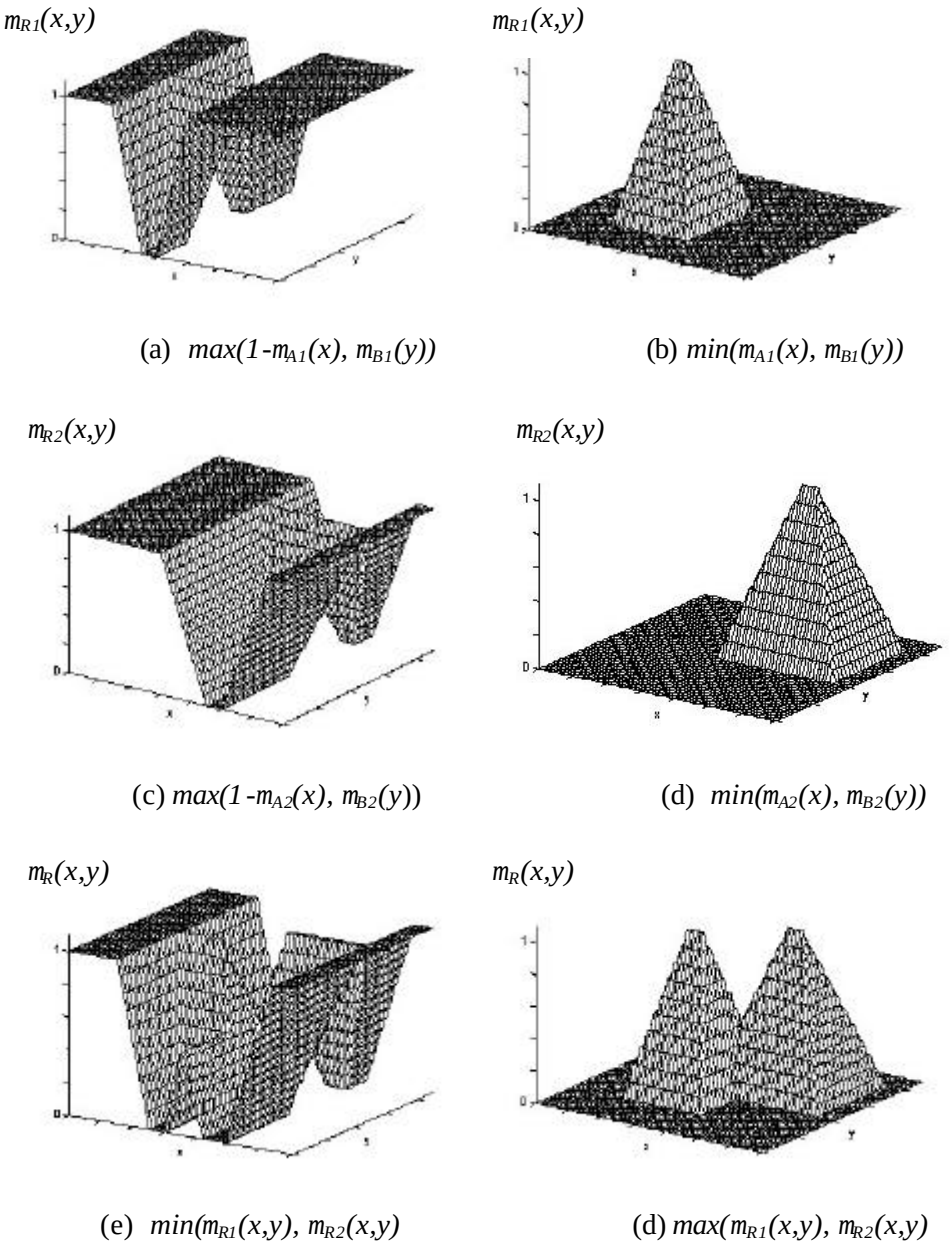



**Figure 2.9** Agrégation des règles floues. La colonne de gauche montre l'agrégation dans le cas d'une implication au sens de Kleene-Dienes; la colonne de droite montre l'agrégation dans le cas d'une implication au sens de Mamdani. (e) et (f) montrent respectivement la relation R obtenue en combinant $R_1$ et $R_2$. $R_1$ et $R_2$ sont les relations correspondantes respectivement aux deux règles $A_1 \rightarrow B_1$ et $A_2 \rightarrow B_2$. $A_1$ ($A_2$) et $B_1$($B_2$) sont respectivement des ensembles flous comme ceux de la figure 2.8a et 2.b.

## 2.9 CONCLUSION

Dans ce chapitre, nous avons présenté et discuté de la contribution de la logique floue dans le développement des systèmes intelligents. En effet, la logique floue permet la représentation et le traitement de connaissances imprécises et incertaines. Ce genre de connaissances est omniprésent dans les problèmes auxquels l'intelligence artificielle est sollicitée pour apporter des solutions satisfaisantes. Ainsi, l'utilisation de la logique floue dans la résolution de ce type de problème s'avère essentielle.

En estimation des coûts de logiciels, la plupart des modèles utilisent la logique classique pour la représentation et le traitement des imprécisions et des incertitudes tout au long du processus de formulation d'une estimation au coût d'un nouveau projet logiciel. Comme nous l'avons discuté dans ce chapitre, la logique floue, en opposition à la logique classique, permet une représentation et un traitement adéquats des imprécisions et incertitudes, a priori similaires à ceux utilisés par les humains, et ceci à travers ses différents concepts tels que les ensembles flous et le raisonnement flou. Ainsi, nous avons opté pour l'utilisation de la logique floue dans le processus d'estimation par analogie afin de lui intégrer les deux premières caractéristiques de l'intelligence, à savoir la gestion des imprécisions et la gestions des incertitudes au niveau des estimations fournies. Cependant, dans un premier temps, nous appliquons la logique floue au cas du modèle COCOMO'81 pour la représentation de ces différents facteurs du coût et ceci pour deux raisons principales: premièrement, nous utiliserons la base de projets COCOMO'81 dans la validation de notre modèle d'estimation par analogie; deuxièmement, nous vérifierons, au préalable, à travers cette application, les avantages de l'utilisation de la logique floue, à la place de la logique classique, dans un processus d'estimation des coûts.

# CHAPITRE III

## REPRÉSENTATION ET TRAITEMENT DES VALEURS LINGUISTIQUES DANS LES MODÈLES D'ESTIMATION DES COÛTS: CAS DU MODÈLE COCOMO'81

L'utilisation des valeurs linguistiques dans l'évaluation de plusieurs attributs d'un logiciel contraigne les modèles d'estimation des coûts. En effet, ces valeurs révèlent l'existence des imprécisions et des incertitudes dans le processus de mesurage de la plupart des attributs d'un logiciel. Ainsi, les modèles d'estimation des coûts doivent prendre en considération ces imprécisions et ces incertitudes dans leur processus de formulation des estimations de coûts. Ce chapitre discute, tout d'abord, de l'utilisation massive des valeurs linguistiques dans l'évaluation des différents attributs d'un logiciel; ensuite, il présente l'application de la logique floue pour la représentation et pour le traitement des valeurs linguistiques utilisées dans le modèle COCOMO'81.

### 3.1 INTRODUCTION

Les modèles paramétriques et non-paramétriques d'estimation des coûts utilisent encore la logique classique pour traiter le cas des projets logiciels décrits par des valeurs linguistiques. Ainsi, ces valeurs sont représentées dans ces modèles par des intervalles classiques ou des par nombres. L'utilisation de cette représentation impose des restrictions majeures sur ces deux types de modèle d'estimation des coûts. Tout d'abord, elle ne permet pas de gérer les imprécisions et les incertitudes dans le processus d'estimation des coûts; de plus, elle n'est pas adaptée dans le cas où la plupart des variables affectant le coût sont qualitatives. Pour remédier à ces limites, nous utilisons la logique floue pour la représentation et le traitement des valeurs linguistiques dans le processus d'estimation des coûts. Dans ce chapitre, nous traitons le cas des modèles paramétriques et plus particulièrement le modèle COCOMO'81.

Ce chapitre est composé de trois sections. Dans la première section, nous présentons les objectifs, l'historique ainsi que quelques éléments de la métrologie de logiciels. Dans la

deuxième section, nous expliquons les raisons qui font de la métrologie de logiciels une science assez complexe. Nous traitons particulièrement le cas de l'utilisation massive des valeurs linguistiques dans le processus de mesurage en génie logiciel. Dans la troisième section, nous présentons l'application de la logique floue au modèle COCOMO'81 afin de représenter et de traiter convenablement les valeurs linguistiques du modèle.

## 3.2 LA MESURE DE LOGICIELS

### 3.2.1 Objectifs

À la fin des années 1960 se tenait une conférence internationale sur le thème de *la crise des logiciels*. Déjà, à cette époque, industriels et informaticiens s'accordaient tous sur le fait que le développement de logiciels coûtait trop cher, que les temps de réalisation étaient trop longs et que la qualité des produits logiciels livrés étaient de plus en plus insatisfaisante. C'est donc pour surmonter ces problèmes que, à l'issue de cette conférence, est apparue la discipline du génie logiciel. Cette discipline s'est attachée, depuis, à fournir des techniques et des outils pour le développement et la maintenance de logiciels. L'objectif principal était d'élever les activités de production des logiciels au rang d'une industrie dont les technologies seraient contrôlées et maîtrisées, afin d'améliorer la qualité des logiciels et de réduire leurs coûts de développement et de maintenance.

Après une dizaine d'années d'expérimentations des nouvelles méthodes du génie logiciel dans le développement et la maintenance des systèmes informatiques, les progrès réalisés en industrie de logiciels étaient insuffisantes et se heurtaient à plusieurs difficultés relevant de la nature du domaine et du manque de procédures pour contrôler et pour évaluer les différents attributs d'un logiciel. Dans les autres domaines tels que le génie électrique, la médecine et le génie civil, les progrès scientifiques dépendaient étroitement de la possibilité de faire des mesures sur les différents attributs étudiés. Il n'est pas étonnant de constater à quel point l'invention du thermomètre à mercure par Fahrenheit, celle du chronomètre par Harrison, et celle des instruments servant à mesurer la pression, la tension électrique, la vitesse, la force de pesanteur et autres phénomènes naturels, apparaissaient comme précurseurs pour certains nouveaux concepts scientifiques et certaines théories solidement fondées.

De cette nécessité, il était indispensable que la communauté du génie logiciel entame des études pour développer des méthodes quantitatives propres à ce domaine afin d'évaluer et de contrôler le développement et la maintenance d'un logiciel. Tom DeMarco, un des pionniers et des défendeurs de la nécessité de la mesure en génie logiciel, confirma cette obligation «You cannot control what you cannot measure » (DeMarco, 1982). En effet, on ne peut contrôler la fiabilité d'un logiciel sans effectuer périodiquement des mesures sur le nombre et la nature des défaillances de ce logiciel. De ce fait, une nouvelle sous-discipline du génie logiciel, à savoir la mesure de logiciels, a été initiée afin d'achever les objectifs visés (maîtriser les coûts, améliorer la qualité, réduire le temps de développement, etc.). En général, la mesure de logiciels sert à atteindre deux objectifs principaux (Fenton et Pfleeger, 1997):

1- *Évaluer pour comprendre:* l'évaluation des attributs d'un logiciel nous permet de comprendre ce qui s'est passé durant le développement et la maintenance de ce logiciel. Ainsi, nous pourrons éviter les erreurs commises et profiter des bénéfices de cette expérience dans un futur projet logiciel. Cela nous permettra d'améliorer les processus de développement et de maintenance afin de produire des logiciels de qualité très élevée avec des coûts raisonnables.

2- *Prédire pour contrôler*: la prédiction des attributs d'un projet logiciel nous permet de contrôler son état d'avancement. Par exemple, les modèles d'estimation des coûts de développement de logiciels nous permettent de prédire le coût d'un logiciel à une étape assez précoce dans son cycle de vie, et ceci afin de bien gérer la répartition du coût estimé sur les différentes phases de développement.

### 3.2.2 Historique

Les premiers travaux sur le développement de mesures en génie logiciel datent de 1975 quand Halstead a publié son livre intitulé *Elements of Software Science* (Halstead 1975). Halstead estime avoir fourni dans ce livre tous les éléments de base pour évaluer les différents attributs d'un logiciel tels que l'effort de développement, la complexité, la durée du codage et la difficulté d'un programme (Sect. 1.5.2). En 1976, McCabe a proposé une mesure

pour évaluer la complexité structurelle du code source d'un logiciel (McCabe, 1976). Cette mesure s'est inspirée des idées et des notations de la théorie des graphes. McCabe a défini la complexité structurelle d'un programme comme étant le nombre cyclomatique du graphe de flux de contrôle correspondant au programme, c'est-à-dire le nombre de régions fermées dans le graphe. En 1979, Albrecht a proposé une méthodologie, les Points de Fonction (FP), pour évaluer la taille d'un logic iel à partir de son dossier de spécification (Albrecht, 1979). Cette approche a pour avantage la possibilité d'être appliquée dès les premières étapes du cycle de développement afin de prédire certains attributs du logiciel tels que l'effort et le temps de développement. En 1981, Henri et Cafura ont proposé une mesure pour la complexité d'un module au niveau de la phase de conception d'un logiciel. Cette mesure détermine la complexité d'un module en fonction de sa longueur, le flot entrant (*Fan-in*), et le flot sortant (*Fan-out*) (Henri et Cafura, 1981).

Cependant, malgré le nombre important de mesures proposées dans la littérature pour évaluer et/ou prédire les différents attributs d'un logiciel, le besoin d'une approche rigoureuse pour définir et pour valider une mesure s'est révélé indispensable au sein de la communauté pour une bonne pratique de la mesure en génie logiciel.

### 3.2.3 Vers une science de la mesure de logiciels

Fenton et Pflegeer définissent les éléments de base de la mesure de logiciels en tant qu'application de la théorie de mesure en génie logiciel (Fenton et Pfleeger, 1997). Ils identifient trois types d'entités et deux types d'attributs en génie logiciel:

- Type des entités logicielles: Processus, Produit ou Ressource. Le tableau 3.1 résume la définition et donne des exemples de chaque type d'entité en génie logiciel.

- Type des attributs de logiciels: un attribut est toujours lié à une entité logicielle. Il peut être Interne ou Externe dépendamment de la façon avec laquelle il est mesuré. Le tableau 3.2 résume la définition et donne des exemples de chaque type d'attributs.

**Tableau 3.1**
Types, définitions et exemples d'entités en génie logiciel

| Entités | Définitions | Exemples |
|---|---|---|
| Processus | Les activités effectuées durant le cycle de vie d'un logiciel | Processus de spécification, de conception, de tests, etc. |
| Produit | Les différentes sorties enregistrées durant le cycle de vie d'un logiciel | Dossier de spécification, code source d'un programme, plan de tests, etc. |
| Ressource | Les entrées d'un processus | Le personnel, les outils, les méthodes, le matériel, etc. |

**Tableau 3.2**
Types, définitions et exemples d'attributs d'un logiciel

| Attributs | Définitions | Exemples |
|---|---|---|
| Interne | Il est mesuré uniquement en termes d'entité à laquelle il est relié et indépendamment de son environnement. | Taille d'un programme, nombre de fautes dans un dossier de spécification, expérience du personnel, etc. |
| Externe | Il est mesuré dépendamment de l'environnement de l'entité à laquelle il est associé. | Productivité du personnel, fiabilité logicielle, qualité logicielle |

Selon Fenton et Pflegeer, une mesure concerne donc un attribut d'une entité logicielle. Par exemple, la mesure de McCabe concerne l'attribut *complexité* de l'entité Produit: *logiciel.* Fenton et Pfleeger définissent une mesure par (Fenton et Pfleeger, 1997):

> «Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.»

À partir de cette définition, nous pouvons distinguer trois étapes dans le processus de mesurage d'un attribut: (1) l'identification de l'attribut à mesurer ainsi que l'entité logicielle à laquelle il est associé; (2) la définition des règles, des hypothèses et des procédures constituant le processus de mesurage; (3) l'affectation d'une valeur à l'attribut (Fig. 3.1).

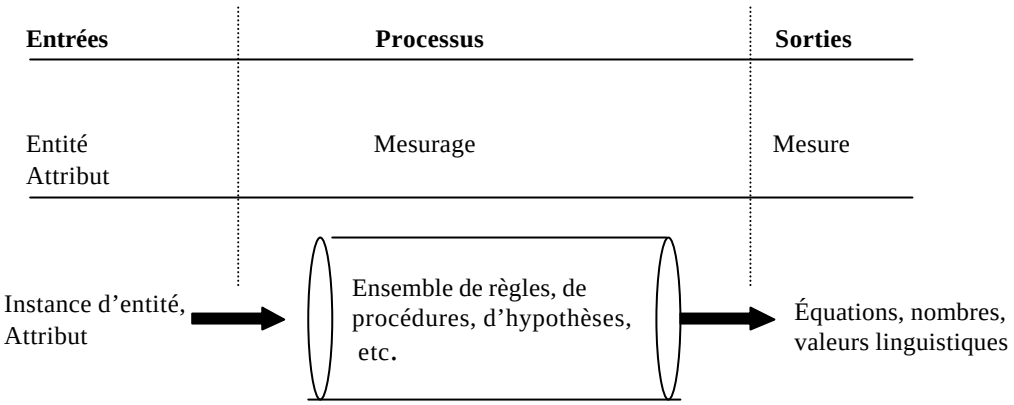



**Figure 3.1** Processus de mesurage d'un attribut d'une entité logicielle.

Du nos jours, les travaux de recherche en mesure de logiciels ne semblent pas encore aboutir aux objectifs escomptés. De plus en plus, la communauté est consciente de la difficulté de la tâche à cause des spécificités du domaine du génie logiciel. Par conséquent, il ne semble pas que le succès, qu'a eu l'application de la théorie de mesure dans les domaines classiques tels que le génie électrique, le génie civil, le génie mécanique et la médecine, se concrétisera facilement en génie logiciel. En effet, en médecine par exemple, on ne se sert pas que de l'âge ou de la taille du patient pour se prononcer sur son état; malheureusement, en génie logiciel, nous en sommes à une étape où l'on a seulement des mesures similaires à l'âge et à la taille d'une personne. Ces mesures sont trop superficielles et ne peuvent être suffisantes pour évaluer et pour contrôler les différents aspects d'un logiciel. Dans la section suivante, nous discutons de quelques raisons qui font de la mesure de logiciels une science assez complexe. Nous traitons particulièrement le cas de l'utilisation des valeurs linguistiques dans le processus de mesurage en génie logiciel.

## 3.3 VALEURS LINGUISTIQUES EN MESURE DE LOGICIELS

En 2003, la théorie de mesure a déjà plus d'un siècle. Selon Zuse, elle a commencé avec l'article de Helmholtz *Counting and Measuring from Epistemological Point of View* et s'est concrétisée avec les travaux de Krantz et al. sur la théorie de la représentation axiomatique

d'une mesure (Krantz et al., 1971; Zuse, 1998). La théorie de mesure consiste en trois volets principaux (Zuse, 1998):

- La détermination des échelles de mesures ainsi que leurs types,
- la discussion des conditions et des pré-requis de l'attribut à mesurer, et
- l'étude des conditions de transformation d'un système de relations empiriques à un système de relations numériques équivalent.

Comme dans d'autres disciplines (génie électrique, génie civil, médecine, etc.), l'application de la théorie de mesure en génie logiciel a été l'objet de plusieurs travaux de recherche durant les trois dernières décades. Ces travaux de recherches consistaient à développer des mesures pour l'évaluation, le contrôle et la prédiction des attributs les plus pertinents d'un logiciel tels que l'effort de développement, la fiabilité logicielle et la productivité du personnel. L'objectif de ces travaux de recherche était l'amélioration du processus de développement et de maintenance de logiciels et par conséquent, l'amélioration de la qualité des différentes produits issus du cycle de vie d'un logiciel (document de spécification, document de conception, code source, plan de tests, etc.).

Cependant, le domaine du génie logiciel présente plusieurs caractéristiques qui font de l'application de la théorie de mesure à ce domaine une tâche très complexe. En effet:

- Le génie logiciel est une science encore *jeune* qui concerne un produit, *le logiciel*, dont les caractéristiques et les attributs diffèrent beaucoup de ceux des autres produits des industries classiques tels que les voitures, les téléviseurs et les caméras. Par exemple, des attributs de logiciels tels que la complexité, la taille et la fiabilité ne sont pas encore clairement définies. Par conséquent, on retrouve dans la littérature, pour le même attribut, une multitude de définitions qui dépendent des besoins de l'environnement impliqué[5].

[5] Comme exemple de ces attributs, nous pouvons citer la complexité logicielle pour laquelle Zuse avait dénombré presque 100 mesures dans son ouvrage *Software Complexity: Measures and Methods* (Zuse, 1991).

- Les humains sont directement impliqués dans le processus de mesurage de presque tous les attributs d'un logiciel. Ainsi, la plupart des attributs d'un logiciel sont qualitatifs plutôt que quantitatifs. Ils sont donc mesurés sur des échelles composées de valeurs linguistiques telles que *très bas*, *bas* et *excellent*. Par exemple, dans le modèle COCOMO'81, 15 parmi 17 facteurs sont mesurés sur une échelle composée de six qualifications (*très faible, faible, moyen, élevé, très élevé* et *extra-élevé*; Boehm, 1981). Aussi, dans le COCOMO II, 22 parmi 24 facteurs sont de type qualitatifs (Boehm et al., 1995; Chulani, 1998). De même, dans la méthode des points de fonctions (FP) mesurant la taille fonctionnelle d'un logiciel, les cinq éléments de base de la méthode (Entrées, Sorties, Questions, Fichiers et Interfaces) sont évaluées sur une échelle composée de trois valeurs linguistiques (*faible, moyen* et *élevé*). Aussi, dans la méthode FP, le calcul du coefficient GSC (*General System Characteristics*), représentant les caractéristiques générales de l'environnement de l'application, prend en considération 14 facteurs qui sont tous mesurés sur une échelle composée de six valeurs linguistiques (*aucune influence, non significatif, moyen, significatif, essentiel*) (Albrecht et Gaffney, 1983).

La deuxième caractéristique concerne les attributs qualitatifs tels que la fiabilité, la maintenabilité et la portabilité. Elle est en relation avec le type d'échelles du fait que celle-ci indique le degré de compréhension de l'attribut étudié. Souvent, en mesure de logiciels, la communauté utilise les cinq types d'échelles définis par Stevens en 1946: Nominale, Ordinale, Intervalle, Ratio et Absolue (Stevens, 1946). Ces types d'échelles sont hiérarchiquement ordonnés de la moins fine à la plus fine. Le type d'échelles est défini par l'ensemble de transformations admissibles qui peuvent être appliquées à cette échelle (Tab. 3.3).

La détermination du type d'échelles est très importante car elle met en évidence les propriétés empiriques associées à l'attribut mesuré. Par exemple, on ne peut avoir une échelle de type Ratio que pour les attributs possédant des relations empiriques qui utilisent des opérations de rapport telles que *2 fois* ou *3 fois*. Le type d'échelles détermine aussi: 1) les opérations statistiques applicables sur les valeurs de l'échelle (par exemple la moyenne arithmétique

nécessite que l'échelle soit au minimum du type Intervalle); 2) si l'échelle a une unité de mesure; et 3) le type des affirmations significatives dans l'échelle.

**Tableau 3.3**
Les types d'échelles

| Type d'échelle | Transformations admissibles, G | Exemples |
|---|---|---|
| Nominale | G est une correspondance 1-1 | Étiquettes |
| Ordinale | G est une fonction monotone croissante ($x \geq y \Rightarrow G(x) \geq G(y)$) | Préférence, difficulté, qualité, intelligence |
| Intervalle | $G(x)=\alpha x+\beta$, $\alpha>0$ | Température (Fahrenheit, Celsius), temps calendaire |
| Ratio | $G(x)=\alpha x$, $\alpha>0$ | Longueur, température (Kelvin) |
| Absolue | G=identité | Comptage d'entités |

Les attributs qualitatifs sont souvent mesurés sur une échelle de type Nominale ou Ordinale. L'échelle de type Nominale représente le niveau le moins fin et ne permet que la classification des différentes entités logicielles dans des catégories différentes. Des exemples de ce type d'échelles, en mesure de logiciels, sont le type de l'application à développer (Finance, Gestion, Commerce, Contrôle, etc.) et le langage de programmation (C, C++, Java, Cobol, etc.). En plus de la classification, un type d'échelles Ordinale permet d'ordonner les différentes catégories utilisées dans la classification. Des exemples d'attributs qui peuvent être mesurés sur un type d'échelles Ordinale sont la complexité logicielle (*simple*, *moyen*, *complexe*) et la qualité logicielle (*très faible*, *faible*, *moyen*, *élevé*, *très élevé*).

En génie logiciel, la plupart des attributs critiques sont mesurés sur une échelle au plus Ordinale. La présence massive des attributs qualitatifs (mesurés sur une échelle Nominale ou Ordinale) est due au fait que les humains sont directement impliqués dans le processus de mesurage de ces attributs. Par conséquent, les résultats des mesures sont influencés par leurs jugements et leurs opinions. Ainsi, les humains utilisent des valeurs linguistiques telles que *très bas*, *bas*, *simple*, et *moyen* plutôt que des valeurs numériques. C'est ce genre de valeurs linguistiques qui est omniprésent dans la plupart des cas de mesure des attributs d'un logiciel.

Cependant, la représentation et le traitement de ces valeurs linguistiques en mesure de logiciels utilisent encore la logique classique. En effet, les valeurs linguistiques sont souvent représentées par des intervalles classiques ou des nombres. Par exemple, dans la méthode des points de fonction, les cinq entités (Entrées, Sorties, Questions, Fichiers et Interfaces) sont évaluées sur une échelle de trois valeurs linguistiques (*simple*, *moyen*, *complexe*). Chacune de ces trois valeurs linguistiques est représentée par un intervalle classique selon le nombre de fichiers et d'items référencés. Le tableau 3.4 montre le cas de l'entité Sorties. Cela signifie qu'une Sortie ne peut être classée que dans une seule catégorie: *simple*, *moyen* ou *complexe*. Par exemple, une Sortie qui référence 25 items et deux fichiers est qualifiée de *complexe*. Par conséquent, le poids qui lui sera associé dans le comptage du nombre FP est 7.

**Tableau 3.4**
Représentation des trois valeurs linguistiques associées à l'entité Sorties de la méthode FP

| | 1à 5 items | 6 à 19 items | Plus de 20 items |
|---|---|---|---|
| 0 ou 1 fichier | Simple (4) | Simple (4) | Moyen (5) |
| 2 ou 3 fichiers | Simple (4) | Moyen (5) | Complexe (7) |
| Plus que 4 fichiers | Moyen (4) | Complexe (7) | Complexe (7) |

Cette représentation des valeurs linguistiques en mesure de logiciels par des intervalles classiques ou des nombres ne permet pas un traitement adéquat des imprécisions et des incertitudes. Comme nous l'avons discuté dans le chapitre précédent, une représentation par des ensembles flous permet de remédier à cette limitation. Dans la section suivante, nous illustrons les avantages d'une telle représentation des valeurs linguistiques dans le cas d'un modèle d'estimation des coûts, à savoir le modèle COCOMO'81.

## 3.4 LA GESTION DES IMPRÉCISIONS ET DES INCERTITUDES DANS LE MODÈLE COCOMO'81

Depuis son apparition, le modèle COCOMO'81 a fait l'objet de plusieurs travaux de calibration, d'amélioration, et de reformulation (Miyazaki et Mori, 1985; Marwane et Mili, 1991; Gulizian, 1991; Amrane et Slimani, 1993; Boehm et al., 1995; Idri, 1997; Idri, Griech et El Iraki, 1997). Dans cette section, nous présentons notre contribution à cette réflexion par l'application du concept de l'ensemble flou au modèle COCOMO'81 Intermédiaire. Cette

application consiste, premièrement, à représenter les valeurs linguistiques du modèle par des ensembles flous plutôt que par des intervalles classiques et, deuxièmement, à reformuler le modèle de telle façon qu'il traite convenablement les imprécisions et les incertitudes dans l'évaluation de ses facteurs du coût.

### 3.4.1 Le modèle COCOMO'81

Le modèle COCOMO'81 est sans doute le modèle le plus connu et le mieux documenté dans toute la littérature d'estimation des coûts. Son auteur Barry Boehm lui a consacré un ouvrage entier: *Software Engineering Economics* (Boehm, 1981). Dans cet ouvrage, une description complète de ce modèle ainsi qu'une discussion de plusieurs aspects touchant au génie logiciel et notamment au domaine des estimations et de conduite de projets logiciels, sont présentées.

COCOMO'81 a été mis au point à partir d'une étude statistique sur une base de données contenant 63 projets logiciels développés aux États-Unis entre les années 1964 et 1979. Il permet de formuler des estimations de l'effort et du temps de développement ainsi que ceux de maintenance d'un logiciel. En plus, il fournit la répartition de cet effort sur les quatre phases du cycle de développement (Analyse, Conception, Codage et Tests unitaires, Intégration et Tests) et sur les huit activités de chaque phase (Analyse de besoins, Conception, Programmation, Plan de test, Vérification et Validation, Gestion du projet, Gestion des configurations et Assurance qualité, Documentation).

Le modèle COCOMO'81 existe en trois versions: Simple, Intermédiaire et Détaillé. La version de base, COCOMO Simple, s'appuie uniquement sur la taille d'un logiciel mesurée en Kilo Instructions Sources Livrées (KISL) et sur le mode du projet qui indique son degré de difficulté. COCOMO'81 distingue trois modes de projets (Boehm, 1981):

- Organique (*Organic*): ce sont les projets relativement simples qui peuvent être réalisés par des équipes de petite taille, travaillant dans un environnement familier et dans un domaine d'application connu par l'équipe. Les membres de l'équipe savent ce qu'ils ont à faire et le feront rapidement.

- Semi-détaché (*Semi-detached*): ce sont les projets qui présentent un degré de difficulté moyen. En effet, l'équipe de développement est constituée de personnes expérimentées et débutantes. Ils ont une expérience limitée du type d'application et du système utilisé pour le développement. L'application est caractérisée par l'existence de quelques interfaces assez complexes.

- Intégré (*Embedded*): ce sont les projets complexes. L'application à développer intègre des contraintes fortes (contraintes de type temps réel, sécurité, support matériel et logiciel complexes). Le coût de changement d'une contrainte est très élevé.

Le COCOMO'81 Simple est peu précis dans ses estimations, mais peut néanmoins être utilisé pour des estimations grossières. Le COCOMO'81 Intermédiaire implique, en plus de la taille et le mode du projet, 15 facteurs qui justifient la variation entre les coûts de deux projets logiciels ayant la même taille. Ces 15 facteurs sont appréciés au niveau de l'ensemble du système. Enfin, le COCOMO'81 Détaillé évalue ces 15 facteurs en fonction des phases et de trois niveaux du logiciel: *système*, *sous-système* et *module*. Par exemple, le facteur PCAP (*Programmer Capability*) n'aura aucun effet sur l'effort durant la phase de conception; par contraste, pendant la phase de codage, il aura des effets positifs si la compétence des programmeurs est *très élevée* et des effets négatifs si elle est *très basse*. L'utilisation du COCOMO'81 Détaillé ne peut être envisagé qu'après avoir mené une conception détaillée dans laquelle le logiciel est décomposé en sous-systèmes et modules. Dans cette thèse, nous nous intéressons au modèle COCOMO Intermédiaire. Le choix de la version intermédiaire du modèle comme sujet de nos analyses est justifié comme suit:

- C'est la version la plus utilisée.

- La précision d'estimation de la version Intermédiaire (MRE=68%), mesurée en termes de l'erreur relative MRE (Équation 1.19), est nettement supérieure à celle de la version Simple (MRE=25%) et se rapproche de celle de la version Détaillée (MRE=70%).

- Les données historiques publiées ne permettent l'évaluation que des versions Simple ou Intermédiaire (Boehm, 1981).

- La version simple ne prend pas en considération assez de facteurs (seulement deux).

### 3.4.2 Le modèle COCOMO'81 Intermédiaire

Le modèle COCOMO'81 Intermédiaire est une extension du modèle COCOMO'81 Simple. Il en diffère principalement par la prise en compte de 15 facteurs, en plus de la taille (mesurée en terme de Kilo Instructions source Livrées) et du mode de logiciel (mesuré sur une échelle de trois valeurs: Organique, Semi-détaché et Intégré), pris comme étant les principaux facteurs affectant le coût de développement. Ces 15 facteurs représentent l'environnement de développement du logiciel. On retrouve, par exemple, la compétence des analystes, l'expérience dans le langage de programmation utilisé et les contraintes sur les délais de développement. Ils sont regroupés en quatre classes (Tab. 3.5). Selon Boehm, ces 15 facteurs justifient la variation entre les coûts de deux logiciels de même taille. Cette variation n'est pas révélée ni expliquée par le COCOMO'81 Simple. L'équation d'estimation du modèle COCOMO'81 Intermédiaire de l'effort de développement en Homme-Mois (HM) d'un projet logiciel est:

$$Effort(HM) = a \times taille^{b} \prod_{i=1}^{15} c_{ij} \quad j = 1..k_i \qquad \text{( Équation 3.1)}$$

où *taille* désigne la taille d'un logiciel mesurée en KISL,

$a$, $b$ des constantes qui dépendent du mode du projet logiciel (Organique, Semi-détaché ou Intégré),

$C_{ij}$ le multiplicateur d'effort correspondant à la $j^{ème}$ catégorie (valeur linguistique) sélectionnée du $i^{ème}$ facteur de coût (Tab. 3.6), et

$k_i$ le nombre de catégories (valeurs linguistiques) correspondant au $i^{ème}$ facteur de coût.

L'effet sur le coût de chacun des 15 facteurs est exprimé en termes de ratio de productivité, qui est le rapport entre la productivité (exprimée en KISL par Homme-Mois) d'un projet exploitant au mieux le facteur en question et la productivité d'un projet n'exploitant pas ce facteur; tous les autres facteurs sont supposés identiques dans les deux projets (Fig. 3.2).

**Tableau 3.5**
Les 15 facteurs du modèle COCOMO'81 Intermédiaire

**Attributs Produit**
RELY (*Required Software Reliability*): Fiabilité requise
DATA (*Data Base Size*): Taille de la base de données
CPLX (*Product Complexity*): Complexité du logiciel

**Attributs Matériel**
TIME (*Execution Time Contraint*): Contrainte du temps d'exécution
STOR (*Main storage Contraint*) Contrainte de la taille mémoire
VIRT (*Virtual Machine Volatility*): Instabilité de la machine virtuelle
TURN (*Computer Turnaround Time*): Temps de restitution des travaux
VEXP (*Virtual Machine Experience*): Expérience dans la machine virtuelle

**Attributs Personnel**
ACAP (*Analyst Capability*): Compétence des analystes
AEXP (*Application Experience*): Expérience du domaine d'application
PCAP (*Programmer Capability*): Compétence des programmeurs
LEXP (*Programming Language Experience*): Expérience du langage de programmation

**Attributs Projet**
MODP (*Modern Programming Practices*): Pratiques des méthodes de programmation
TOOL (*Use of Software Tools*): Utilisation d'outils logiciels
SCED (*Required Development Schedule*): Contraintes de plannification

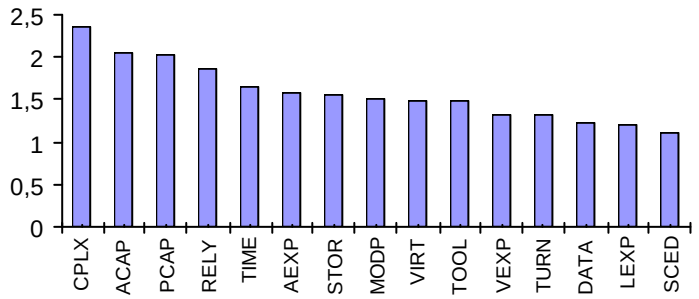



**Figure 3.2** Comparaison des ratios de productivité des 15 facteurs.

**Tableau 3.6**
Les 69 multiplicateurs de l'effort $C_{ij}$, associés aux 15 facteurs du COCOMO'81 Intermédiaire

| Attribut | Valeurs linguistiques | | | | | |
|---|---|---|---|---|---|---|
| | *Très bas* | Bas | Moyen | Élevé | Très élevé | Extra-élevé |
| RELY | 0,75 | 0,88 | 1,00 | 1,15 | 1,40 | |
| DATA | | 0,94 | 1,00 | 1,08 | 1,16 | |
| CPLX | 0,70 | 0,85 | 1,00 | 1,15 | 1,30 | 1,65 |
| TIME | | | 1,00 | 1,11 | 1,30 | 1,66 |
| STOR | | | 1,00 | 1,06 | 1,21 | 1,56 |
| VIRT | | 0,87 | 1,00 | 1,15 | 1,30 | |
| TURN | | 0,87 | 1,00 | 1,07 | 1,15 | |
| ACAP | 1,46 | 1,19 | 1,00 | 0,86 | 0,71 | |
| AEXP | 1,29 | 1,13 | 1,00 | 0,91 | 0,82 | |
| PCAP | 1,42 | 1,17 | 1,00 | 0,86 | 0,70 | |
| VEXP | 1,21 | 1,10 | 1,00 | 0,90 | | |
| LEXP | 1,14 | 1,07 | 1,00 | 0,95 | | |
| MODP | 1,24 | 1,10 | 1,00 | 0,91 | 0,82 | |
| TOOL | 1,24 | 1,10 | 1,00 | 0,91 | 0,83 | |
| SCED | 1,23 | 1,08 | 1,00 | 1,04 | 1,10 | |

### 3.4.3 Application de la logique floue au modèle COCOMO'81 Intermédiaire

Dans le modèle COCOMO'81 Intermédiaire, 15 facteurs (Tab. 3.6) parmi 17 sont mesurés sur une échelle de type Ordinale. Cette échelle est composée de six valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé*, *très élevé*, et *extra-élevé*. Ces six valeurs sont représentées, dans le modèle, par des intervalles classiques (voir Annexe A). Par exemple, le facteur DATA, expliquant l'influence de la taille de la base de données sur le coût du logiciel, est mesuré par le ratio:

$$\frac{D}{P} = \frac{Database\,size\,in\,bytes\,or\,characters}{Software\,product\,size\,in\,DSI}$$

Ensuite, une valeur linguistique est attribuée à DATA en fonction du ratio D/P, tel qu'illustré au tableau 3.7.

**Tableau 3.7**
Valeurs linguistiques du facteur DATA

| | Bas | Moyen | Élevé | Très élevé |
|---|---|---|---|---|
| D/P | D/P < 10 | 10≤ D/P < 100 | 100 ≤D/P < 1000 | D/P ≥ 1000 |

Par conséquent, chaque projet logiciel appartient à une et une seule classe. Par exemple, si D/P est égal à 9,99 le DATA du projet logiciel sera alors déclaré *bas*; si D/P est égal à 10,01 il sera alors *moyen*. Cela pose de sérieux problèmes pour les projets ayant des ratios D/P au voisinage des limites. Supposons que nous ayons deux projets logiciels $P_1$ et $P_2$ tels que:

- Pour tous les facteurs ayant une influence croissante sur le coût (RELY, DATA, CPLX, TIME, STORE, VIRT, TURN), $P_1$ est juste avant la borne inférieure de l'intervalle représentant la valeur *élevé*, $P_2$ est juste après.

- Pour les facteurs ayant une influence décroissante sur le coût (ACAP, AEXP, PCAP, VEXP, LEXP, MODP, TOOL), $P_1$ est juste après la borne inférieure de l'intervalle représentant la valeur *moyen*, $P_2$ est juste après.

Dans ce cas, si les deux projets logiciels $P_1$ et $P_2$ ont le même effort nominale ($Effort_{nom} = A \acute{} size^B$), 15H-M, alors l'effort ajusté (Équation 3.1) de $P_1$ est inchangé, tandis que celui de $P_2$ est 52H-M.

Pour remédier à ce problème, dans le cas du modèle COCOMO'81 Intermédiaire, nous proposons de représenter les valeurs linguistiques utilisées dans le modèle par des ensembles flous plutôt que par des ensembles classiques. Les avantages de cette représentation sont:

- Elle est plus générale.
- Elle imite la manière avec laquelle les humains interprètent les valeurs linguistiques.
- La transition d'une valeur linguistique à la valeur linguistique suivante est progressive plutôt que brusque.

Par exemple, pour le cas du facteur DATA, nous avons défini pour chaque valeur linguistique un ensemble flou avec une fonction d'appartenance μ de form trapezoîdale (Fig. 3.3).

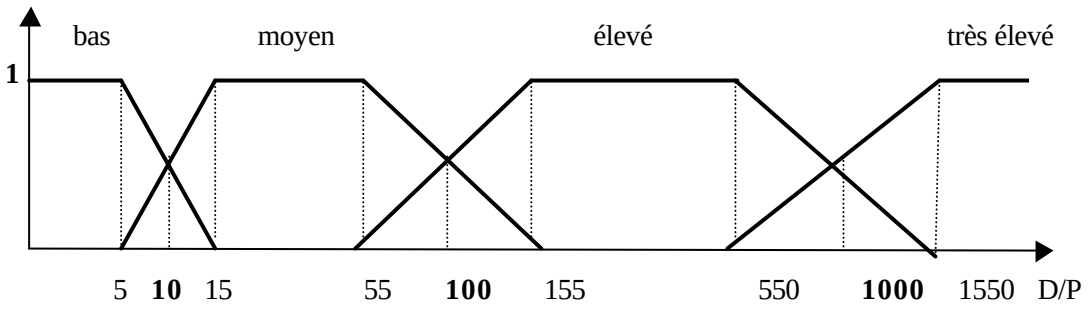



**Figure 3.3** Les fonctions d'appartenance des différents ensembles flous définis pour les quatre valeurs linguistiques du facteur DATA.

Nous avons aussi défini les ensembles flous correspondant aux autres facteurs du modèle COCOMO'81 Intermédiaire (voir Annexe A). Parmi ces 15 facteurs du coût, les quatre attributs RELY, CPLX, MODP, et TOOL n'ont pas été représentés par des ensembles flous car leurs descriptions sont insuffisantes dans (Boehm, 81).

### 3.4.4 Évaluation de la précision du *fuzzy* COCOMO'81 Intermédiaire

Dans cette section, nous évaluons la précision du modèle COCOMO'81 Intermédiaire en utilisant l'équation (3.1) et les nouveaux multiplicateurs d'effort ($F\_C_{ij}$) obtenus à partir des ensembles flous associés aux facteurs affectant le coût (*fuzzy* COCOMO'81 Intermédiaire). Les multiplicateurs d'effort $F\_C_{ij}$, sont calculés à partir des multiplicateurs d'effort classiques (Tab. 3.6) et des fonctions d'appartenance μ associées aux différents ensembles flous définis pour les facteurs du modèle:

$$F\_C_{ij} = F(m_{A_1^i}(P).....m_{A_j^i}(P), C_{i1}.....C_{ij}) \qquad \text{(Équation 3.2)}$$

La détermination de la fonction $F$ nécessite une étude très approfondie. Elle peut être linéaire, polynomiale, exponentielle ou autre. Pour simplifier, nous considérons que $F$ est une fonction linéaire:

$$F\_C_{ij} = \sum_{j=1}^{k_i} m_{A_j^i}(P) \times C_{ij} \qquad \text{(Équation 3.3)}$$

où $m_{A_j^i}$ est la fonction d'appartenance à l'ensemble flou $A_j^i$.

Cette évaluation consiste à comparer les estimations du *fuzzy* COCOMO'81 Intermédiaire avec celles de la version classique du COCOMO'81 Intermédiaire. Dans les deux cas, la précision des estimations est évaluée par l'erreur relative (MRE), le niveau de prédiction (Pred (p)), le minimum des MREs (Min MRE), le maximum des MREs (Max MRE), la moyenne des MREs (MMRE) et l'écart-type des MREs (SDMRE). Dans la littérature, on considère souvent p égal à 0,25 pour l'indicateur Pred(p); cependant, dans cette évaluation, nous fixons p à la valeur 0,20 car c'est cette valeur qui a été utilisée par Boehm dans l'évaluation de la version originale du COCOMO'81. Ainsi, Pred (0,20) représente le nombre de projets logiciels ayant un MRE inférieur ou égal à 0,20.

L'évaluation se fera sur trois bases de données ($DB_1$, $DB_2$ et $DB_3$) déduites en utilisant une procédure aléatoire à partir de la base de données originale du modèle COCOMO'81. En effet, cette dernière contient seulement les multiplicateurs d'effort associés aux 63 projets du modèle; or, pour calculer les $m_{A_j^i}(P)$ de l'équation 3.2, nous aurons besoin des valeurs réelles des différents facteurs des 63 projets. Ainsi, chacune des trois bases contient 63 projets décrits par des valeurs générées aléatoirement selon les qualifications données de leurs facteurs dans la base originale du modèle. Par exemple, le facteur DATA du cinquième projet de la base de données du modèle est déclaré *bas*. Par conséquent, selon le tableau 3.7, les trois valeurs générées pour les trois bases sont entre 0 et 10. Avec ces trois bases ainsi que la base originale de Boehm, nous avons donc quatre bases de données pour effectuer des analyses comparatives entre les performances de la version classique et celles de la version floue du COCOMO'81 Intermédiaire. Le tableau 3.8 montre les résultats de l'évaluation, selon les critères cités ci-dessus, de l'utilisation de la technique *fuzzy* par rapport à l'utilisation de la technique classique avec chacune des quatre bases de données. Dans chaque colonne relative à une base de données ($DB_1$, $DB_2$ et $DB_3$), le résultat de gauche provient de la technique *fuzzy* et celui de droite de la technique classique.

Les résultats de l'évaluation du *fuzzy* COCOMO'81 Intermédiaire sont différents pour les quatre bases de données contrairement au cas de la version classique qui donne les mêmes degrés de précision pour les quatre bases. Cela implique que:

- Le *fuzzy* COCOMO'81 Intermédiaire tolère les imprécisions au niveau de ses entrées et génère en conséquence des sorties (effort) progressivement différentes (autant que la différence entre les entrées est grande autant celle entre les sorties est grande). Cette progression permet au modèle d'être moins sensible à des petites variations dans ses entrées. Cette stabilité du modèle est l'un des dix critères cités par Boehm que n'importe quel modèle d'estimation des coûts doit satisfaire (voir Sect. 1.7).

- La version originale du COCOMO'81 Intermédiaire ne tolère pas les imprécisions au niveau de ses entrées et donne, soit les mêmes sorties pour toutes les valeurs à l'intérieur des mêmes intervalles (Tab. 3.8), soit des sorties significativement différentes lorsqu'il y a passage d'un intervalle à l'autre, même si les changement de valeurs sont très petits (cas de l'exemple de la section précédente). Par conséquent, elle est moins stable et très sensible aux différentes variations dans ses entrées.

**Tableau 3.8**
Résultats de l'évaluation du *fuzzy* vs original COCOMO'81 Intermédiaire

| | *fuzzy* vs original COCOMO'81 Intermédiaire | | | | | |
|---|---|---|---|---|---|---|
| | **$DB_1$** | | **$DB_2$** | | **$DB_3$** | |
| | *fuzzy* | original | *fuzzy* | original | *fuzzy* | original |
| Pred(20) (%) | 62,14 | 68 | 46,86 | 68 | 41,27 | 68 |
| Min MRE (%) | 0,11 | 0,02 | 0,40 | 0,02 | 0,06 | 0,02 |
| Max MRE (%) | 88,60 | 83,58 | 3233,03 | 83,58 | 88,03 | 83,58 |
| $MMRE_i$ (%) | 22,50 | 18,52 | 78,45 | 18,52 | 30,80 | 18,52 |
| SDMRE [6] | 19,69 | 16,97 | 404,40 | 16,97 | 22,95 | 16,97 |

[6] Standard Déviation des MRE.

Cependant, les résultats obtenus dans le cas de la base de données $DB_2$ pour *fuzzy* COCOMO'81 intermédiaire, spécifiquement l'indicateur Max MRE qui est de l'ordre de 3233,03%, semblent exprimer l'inverse, c'est-à-dire, le *fuzzy* COCOMO'81 intermédiaire est moins sensible aux variations dans ses entrées que le COCOMO'81 classique du fait que ce dernier génère les mêmes estimations pour des projets situés dans le même intervalle. Ceci n'est pas vrai. En effet, ce genre de situations, où les imprécisions des estimations sont assez grandes, se présente dans le cas des projets dits extrêmes (outliers). Dans notre expérimentation, c'était le cas d'un projet de la base de données $DB_2$ pour lequel la plupart des valeurs générées se situaient aux limites des intersections des fonctions d'appartenance représentant les valeurs linguistiques du modèle. Ainsi, en absence du coût réel de ce projet, la comparaison de son coût estimé, par *fuzzy* COCOMO'81 intermédiaire, au coût réel d'un projet totalement différent a généré un MRE assez grande. Ceci est dit, la communauté en estimation des coûts utilise en général l'indicateur Pred(20), du fait que celui-ci est moins sensible aux estimations extrêmes, pour évaluer la sensibilité et/ou la précision d'un modèle. Nous remarquons, d'ailleurs, que les variations du Pred(20) sont acceptables et que le Pred(20) associé à la base $DB_2$ est meilleur que celui associé à la base $DB_3$ bien que le *fuzzy* COCOMO'81 apparaissait moins sensible dans le cas de la base $DB_3$ que dans celui du $DB_2$.

## 3.5 DISCUSSION ET CONCLUSION

Dans ce chapitre, nous avons discuté de l'utilisation des valeurs linguistiques dans le processus de mesurage des attributs d'un logiciel. Nous avons relevé que la représentation et le traitement de ces valeurs se font selon la logique classique. Or, nous avons déjà discuté, dans le chapitre précédent, des avantages de l'utilisation de la logique floue pour la représentation et le traitement de ce genre de valeurs. Ainsi, nous avons étudié le cas des modèles d'estimation des coûts. Afin de permettre à ces modèles de mieux gérer les imprécisions et les incertitudes rattachées souvent aux valeurs linguistiques, nous avons proposé l'utilisation de la logique floue pour leur représentation et leur traitement. Nous avons illustré cette proposition en appliquant la logique floue au modèle COCOMO'81 Intermédiaire. Ainsi, nous avons utilisé les ensembles flous plutôt que les ensembles classiques pour représenter les différentes valeurs linguistiques du modèle COCOMO'81 Intermédiaire. Pour chaque facteur du modèle, nous avons défini les ensembles flous correspondant à ses valeurs linguistiques. Ces ensembles flous ont été représentés par des fonctions d'appartenance trapézoïdales. Les résultats de notre évaluation sont affectés par ce choix. Cependant, il existe d'autres représentations possibles d'un ensemble flou qui pourraient être utilisées (représentation gaussienne, Bell, etc.). Pour choisir la meilleure représentation, il faudrait étudier la signification de chaque valeur linguistique dans l'environnement à partir duquel le modèle COCOMO'81 Intermédiaire a été mis au point. Cette étude s'applique aussi au modèle COCOMO II où 22 facteurs parmi 24 sont mesurés sur une échelle ordinale. Toutefois, parce que les données de ce modèle ne sont pas encore publiées, il n'est pas encore possible de traiter le cas du COCOMO II.

Suite aux résultats encourageants de l'utilisation de la logique floue pour la représentation et pour le traitement des imprécisions dans le modèle COCOMO'81 Intermédiaire, nous envisageons de l'utiliser pour le même objectif dans l'estimation des coûts par analogie. Nous abordons donc l'intégration de notre premier critère d'intelligence: la tolérance des imprécisions dans notre modèle d'estimation des coûts par analogie.

# CHAPITRE IV

# UNE NOUVELLE APPROCHE D'ESTIMATION DES COÛTS PAR ANALOGIE: *FUZZY ANALOGY*

Ce chapitre présente, dans un premier temps, l'état de l'art sur l'utilisation du raisonnement à base de cas (*Case-Based Reasoning*) en estimation des coûts de développement de logiciels. Ainsi, nous identifions deux limitations majeures relatives à l'application de la technique CBR pour formuler une estimation du coût d'un projet logiciel. Premièrement, cette technique ne tolère pas les imprécisions au niveau de la description des projets logiciels ainsi qu'au niveau de l'évaluation de la similarité entre le nouveau projet, pour lequel on veut estimer le coût, et les projets historiques déjà achevés. Deuxièmement, elle ne permet pas la gestion des incertitudes au niveau du coût estimé pour un nouveau projet logiciel.

Pour remédier à ces deux limitations, nous proposons une version floue de la technique CBR pour l'estimation des coûts, *Fuzzy Analogy*. Ce chapitre décrira les trois étapes composant le processus de notre approche *Fuzzy Analogy*: Identification des projets logiciels, Évaluation de la similarité entre les projets logiciels et Adaptation. Nous présentons aussi les résultats de la validation de *Fuzzy Analogy* sur la base de projets COCOMO'81.

## 4.1 RAISONNEMENT PAR ANALOGIE (CBR)

Du fait que le raisonnement est couramment présent dans notre vie quotidienne et constitue la base de la plupart de nos activités mentales, l'intelligence artificielle lui a accordé, tout comme la représentation de la connaissance, un intérêt particulier. D'ailleurs, il y a une étroite imbrication entre le raisonnement et la connaissance: la plupart des mécanismes ou des modèles de raisonnement ne peuvent faire abstraction des connaissances sur lesquelles ils opèrent. Il y a une grande diversité de modes de raisonnement en intelligence artificielle à l'image de la richesse des mécanismes mentaux des humains. Cependant, ils peuvent être regroupés dans deux catégories de raisonnement (Kurtz, Gentner et Gunn, 1999):

- le raisonnement *déductif* dont les conclusions sont obtenues sur des prémisses par l'application d'un formalisme logique, tels que le *modus ponens* et *le modus tollens*, Ce raisonnement opère en *top-down*, c'est-à-dire à partir de cas abstraits, déduisant des cas spécifiques. Par conséquent, ses conclusions sont toujours valides.

- le raisonnement *inductif* dont les inférences sont estimées à partir d'un ensemble d'évidences. Ce raisonnement opère en *bottom-up*. Ainsi, à partir de cas spécifiques, il génère des abstractions: c'est le principe de généralisation. Par conséquent, ses conclusions ne sont pas toujours valides.

L'analogie offre un raisonnement de type inductif puisque à partir de situations similaires, elle génère des comportements similaires. Elle est une forme de raisonnement très courante et efficace chez les humains. Elle guide implicitement des problèmes cognitifs importants et encore mal connus; par exemple l'accès associatif à des mémoires ou la pertinence de ressemblances détectées entre des objets ou des situations. Le paradigme de base du raisonnement par analogie consiste en la mise en correspondance de deux situations, appartenant parfois à deux domaines différents, afin de fournir un comportement adéquat face à une situation nouvelle en fonction d'informations mémorisées sur des situations déjà rencontrées (Gentner, 1998; Gentner, Holoak et Kokinov, 2001; Holyoak et Taghard, 1995). Holyoak et Thagard rapportent que, avant d'être utilisé dans le développement des théories scientifiques, le raisonnement par analogie a servi à la plupart des religions pour expliquer certains concepts de base tels la création, la vie, la mort et la relation entre Dieu et les humains. Souvent, ce genre de concepts n'est pas facile à expliquer directement. Par contraste, ils deviennent compréhensibles quand ils sont expliqués par analogie avec d'autres concepts facilement interprétables. Par exemple, l'analogie suivante *God is related to people as a father is to his children: as progenitor, protector, and disciplinarian* met en évidence la relation entre *God* et *People* à travers celle entre *Father* et *Children* (*R[God, People]=R[Father, Children]*).

Dans le développement des théories scientifiques, Holyoak et Taghard repportent que l'utilisation de l'analogie dans les sciences date d'au moins vingt siècles; la première utilisation reconnue de l'analogie était celle de Vitrivius qui avait décrit la propagation du

son dans la construction des théâtres romains par la forme des vagues (Holyoak et Tagard, 1995). Bien que Holyoak et Thagard reconnaissent que le raisonnement par analogie ne soit pas le seul mécanisme responsable de la créativité chez les humains, ils affirment, par contraste, que c'est le plus important. En effet, l'analogie peut être un moyen efficace pour le développement de nouvelles idées. D'ailleurs, les deux approches dominantes en sciences cognitives sont nées respectivement de l'analogie avec le traitement de l'information dans l'ordinateur (*computing overs symbols*) et de celle avec la structure du cerveau humain (*Brain-like computation*).

Le raisonnement par analogie en IA n'a pas eu le même intérêt que celui accordé aux autres formes de raisonnement telles que le raisonnement formel, le raisonnement par réseaux de neurones et celui par les arbres de décision. En effet, il n'a pas été formalisé rigoureusement et par conséquent, il est resté marginal. Ce n'est que récemment qu'une version simplifiée du raisonnement par analogie a été mise au point sous forme d'un raisonnement à base de cas (*Case-Based Reasoning* -CBR). En effet, le raisonnement à base de cas se limite seulement aux problèmes d'un même domaine alors que le raisonnement par analogie peut être utilisé pour résoudre un problème relevant d'un domaine en se référant à la solution d'un autre problème similaire de domaine différent. La technique CBR se base sur l'hypothèse que la résolution d'un problème consiste en l'accès à des informations mémorisées lors d'expériences précédentes en vue d'une exploitation ultérieure. Elle se caractérise donc par le fait qu'elle utilise des informations spécifiques à des problèmes déjà résolus (des cas) plutôt que des informations générales sur le domaine du problème. Elle retrouve ses origines dans les travaux de Schank sur la mémoire dynamique et ceux de Gentner sur le raisonnement par analogie (Schank, 1982; Gentner, 1983). Depuis, la technique CBR a été développée simultanément par plusieurs groupes de recherche en Amérique et en Europe (Aamodt et Plaza, 1994; Aha, 1991; Kolodner, 1993; Leake, 1996). En général, le processus de la technique CBR est composé de quatre étapes (Aamodt et Plaza, 1994):

1- Retrouver les cas similaires au cas étudié;
2- Utiliser les cas similaires retrouvés pour proposer une solution au problème;
3- Réviser la solution proposée;

4- Retenir les éléments de cette expérience (cas + solution) qui serviront à la résolution des problèmes futurs.

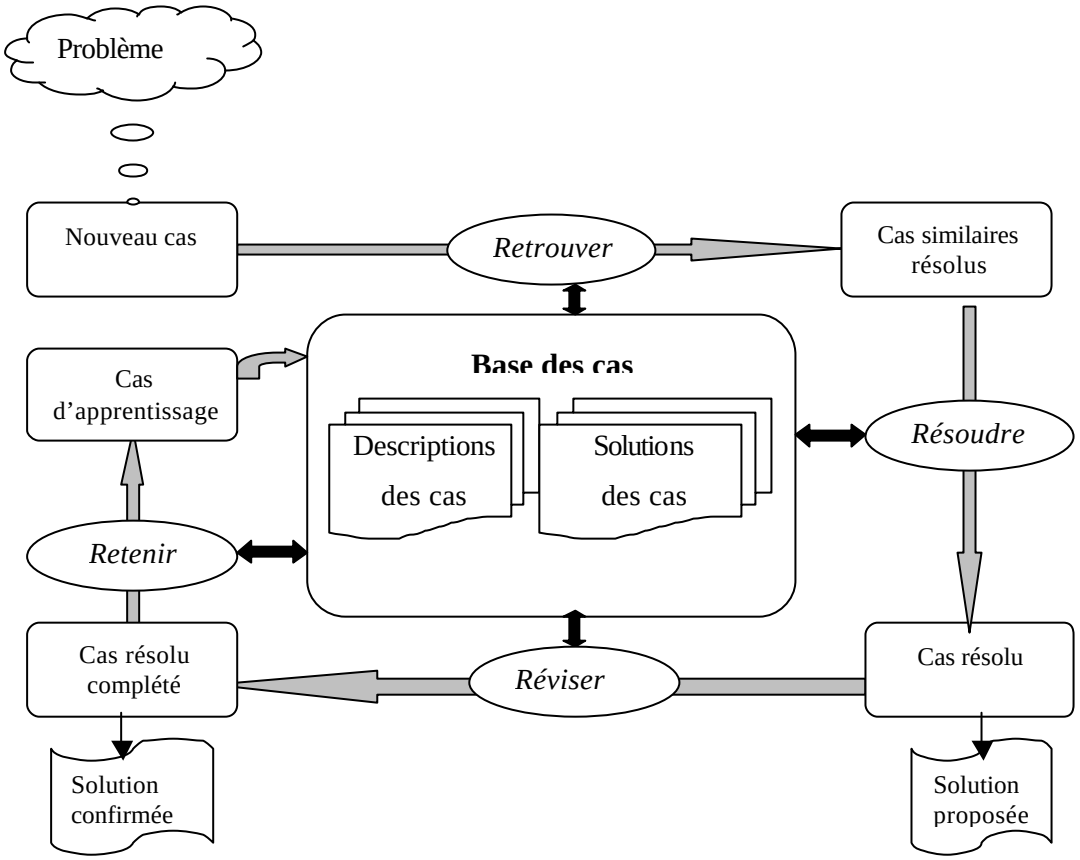



**Figure 4.1** Processus de la technique CBR (Aamodt et Plaza, 1994).

Le processus de résolution d'un problème par la technique CBR commence par la description et par la représentation de ce problème en tant qu'un nouveau cas (Fig. 4.1). En général, un cas est représenté par une structure décrivant son contenu et sauvegardé de façon à le retrouver facilement dans la phase de recherche des cas similaires déjà résolus: phase *Retrouver*. Cette phase consiste à évaluer la similarité entre le nouveau cas et tous les cas déjà résolus (base de cas) en utilisant des mesures de similarité prédéfinies. Ensuite, on choisit le(s) cas le(s) plus similaire(s) au nouveau cas. Ce sont ces cas qui seront utilisés dans la phase *Résoudre* pour proposer une solution au nouveau problème (solution proposée). Cette

solution pourra être, par exemple, identique à celle adoptée dans la situation du cas le plus similaire au nouveau cas ou bien une solution modifiée dépendamment des différences entre les deux cas et des connaissances générales sur le domaine du problème. La phase *Réviser* consiste à évaluer la solution proposée émanant de la phase *Résoudre* et à lui apporter, quand c'est nécessaire, les modifications adéquates afin qu'elle soit adaptée au problème traité. L'évaluation de la solution proposée se fait en l'appliquant concrètement au problème dans le monde réel et en examinant les résultats obtenus par cette application. Si ces résultats ne sont pas satisfaisants, les causes doivent être identifiées afin de modifier la solution précédemment proposée pour qu'elle réponde, la prochaine fois, aux critères d'évaluation (solution confirmée). La dernière phase *Retenir* consiste à utiliser les résultats des trois phases précédentes comme des sources d'apprentissage dans la technique CBR. En effet, son rôle est de décider des modifications qu'il faut apporter à la base de cas afin d'améliorer la performance du processus CBR. Par exemple: Quels sont les éléments pertinents à retenir du nouveau cas résolu afin de les ajouter dans la base de cas? Quel est l'impact de cet ajout sur les autres cas déjà présents dans la base?

Dans la mise au point de notre modèle intelligent d'estimation des coûts, nous nous sommes basés sur la technique CBR pour modéliser la relation entre le coût et les facteurs l'affectant. Ainsi, notre modèle d'estimation adopte le raisonnement à base de cas pour estimer le coût d'un projet logiciel à partir d'un historique de projets logiciels déjà achevés (leurs coûts réels sont connus). En effet, la technique CBR présente, en plus de sa plausibilité psychologique (car elle est une forme simplifiée du raisonnement par analogie), plusieurs avantages sur les autres techniques couramment utilisées en estimation des coûts:

1- Elle est facile à expliquer contrairement au cas des réseaux de neurones.

2- Elle peut modéliser les relations complexes existantes entre le coût et les facteurs l'affectant, contrairement au cas des méthodes statistiques (régression linéaire, par exemple).

3- Elle peut être utilisée dans le cas où on a peu de connaissances sur le problème étudié contrairement au cas des systèmes à base de règles.

4- Elle utilise la connaissance dans sa forme *brute* contrairement au cas des systèmes à base de règles où la connaissance doit être encodée dans des règles (ce processus d'encodage n'est pas souvent facile).

5- Elle peut être utilisée dans le cas où une solution optimale du problème n'est pas évidente.

6- Elle généralise plusieurs types de modélisation en intelligence artificielle:
   a. Cas d'un arbre de régression
   b. Cas de certains réseaux de neurones (Perceptron simple, Réseau de Kohonen, *Radial Basis Function Network*, Réseau de Hopfield) (Sect. 5.5).

## 4.2 ÉTAT DE L'ART SUR L'UTILISATION DU RAIONNEMENT PAR ANALOGIE EN ESTIMATION DES COÛTS

L'idée de l'utilisation de la technique CBR en estimation des coûts n'est pas nouvelle. Boehm avait déjà cité, dans son ouvrage *Software Engineering Economics*, l'estimation par analogie comme étant une technique plausible pour la prédiction des coûts de développement d'un logiciel (Boehm, 1981). Cependant, Boehm n'avait pas proposé une version formelle de l'estimation par analogie. En 1990, Vicinanza, Prietula et Mukhopadhyay ont proposé une reformulation du processus de la technique CBR afin d'être appliqué en estimation des coûts. Cette reformulation est basée sur l'affirmation suivante:

*Similar Software Projects have similar costs.*

Cette affirmation a été déployée comme suit en estimation des coûts: premièrement, chaque projet logiciel (nouveau ou historique) est décrit par un ensemble de caractéristiques telles que la fiabilité et la complexité logicielle, la compétence des analystes intervenant dans le projet et les méthodes de développement utilisées. Ces caractéristiques doivent être indépendantes et significatives pour l'estimation des coûts. Deuxièmement, il faut déterminer la similarité entre le nouveau projet logiciel, auquel on veut estimer le coût, et chaque projet logiciel de la base de projets historiques. Troisièmement, on utilise les coûts des projets logiciels historiques qui sont les plus similaires au nouveau projet pour en déduire une

estimation à son coût. Cette troisième étape est souvent référée dans la littérature par l'étape d'Adaptation.

Vicinanza, Prietula et Mukhopadhyay ont développé un système CBR nommé ESTOR, qui utilise des mesures de similarité (généralement basées sur la taille du logiciel évaluée en LOC ou en FP) spécifiques au domaine d'estimation des coûts pour évaluer la similarité entre les projets logiciels (étape *Retrouver*). Dans l'étape d'Adaptation (étape *Résoudre* dans le cycle classique CBR), ESTOR utilise une base de règles déduites de l'expertise acquise dans l'environnement étudié. Les résultats obtenus lors de cette expérimentation ont été encourageants (Vicinanza, Prietula et Mukhopadhyay, 1990). En effet, l'évaluation de la précision des estimations fournies par ESTOR sur la base de projets de Kemerer indique que ses estimations sont meilleures que celles du modèle COCOMO'81 ainsi que celles des modèles basés sur les points de fonction (Boehm, 1981; Albrecht et Gaffney, 1983; Kemerer, 1987). Cependant, ESTOR présente une limitation majeure au niveau de l'étape d'Adaptation puisqu'il fait appel à une base de règles empiriques très spécifiques à la base de projets de Kemerer. Ainsi, cette base de règles ne peut être utilisée dans les cas d'autres bases de projets logiciels vu les différences existantes entre les environnements de développement de logiciels.

Suite aux résultats encourageants obtenus par Vicinanza, Prietula et Mukhopadhyay, la technique CBR a été l'objectif de plusieurs travaux d'amélioration et de reformulation afin de l'adapter aux spécificités de l'estimation des coûts des projets logiciels (Bisio et Malabocchia, 1995; Shepperd et Schofield, 1997; Kadoda et al., 2000; Angelis et Stamelos, 2000). Shepperd et al. sont parmi les plus impliqués dans le développement d'une technique d'estimation des coûts par analogie (Shepperd, Schofield, et Kitchenham, 1996; Shepperd et Schofiel, 1997; Kadoda et al. 2000). Dans leur évaluation de la performance de la technique CBR, Shepperd et al. discutent des raisons qui ont souvent mené à des résultats différents (Kadoda et al., 2000):

- Celles relevant des caractéristiques de la base de projets logiciels utilisée pour l'évaluation de la performance de la technique CBR (nombre de projets, nombre d'attributs décrivant les projets, types de ces attributs, etc.), et

- celles associées aux différents choix des paramètres de la technique CBR tels que les mesures de similarité utilisées, le nombre de projets logiciels à prendre en considération ainsi que la formule utilisée dans l'étape d'Adaptation.

Ils évaluent les impacts de ces paramètres en utilisant une base de projets logiciels collectés de différents organismes canadiens (Desharnais, 1989). Ils trouvent que le choix du nombre de projets similaires considérés dans l'étape d'Adaptation est important. En effet, le choix des trois premiers projets conduit souvent à des estimations acceptables. Aussi, la stratégie de choix d'un nombre fixe de projets logiciels semble être plus appropriée dans le cas de bases de données contenant un nombre élevé de projets logiciels; par contraste, celle se basant sur un seuil de similarité semble être plus appropriée dans le cas de bases de données contenant un nombre limité de projets logiciels. Enfin, Shepperd et al. estiment, à travers leur expérimentation, que la formule d'Adaptation utilisée n'a pas un effet important sur la précision des estimations fournies par la technique CBR.

Angelis et Stamelos ont étudié l'utilisation de l'estimation par analogie dans le cas de la base de projets d'Ablrecht (Albrecht, 1983). Leur objectif était d'examiner les impacts des choix relatifs à trois paramètres de la technique CBR afin d'offrir une meilleure utilisation ultérieure pour estimer le coût d'un nouveau projet logiciel (Angelis et Stamelos, 2001). Ces trois paramètres sont:

- La mesure utilisée pour évaluer la similarité entre les projets logiciels,
- le nombre de projets similaires à prendre en considération dans l'étape d'Adaptation, et
- la formule statistique utilisée pour le calcul du coût estimé à partir des coûts réels des projets les plus similaires au nouveau projet.

Angelis et Stamelos suggèrent l'utilisation de la méthode *Bootstrapping* pour le choix et la configuration de ces trois paramètres. Cette méthode consiste en la construction de nouveaux échantillonnages de projets logiciels à partir de la base originale de projets d'Albrecht. Ce

sont ces échantillonnages qui seront utilisés pour le choix des valeurs des trois paramètres qui fournissent les meilleures précisions d'estimation des coûts.

L'estimation par analogie est une alternative prometteuse qui s'adapte bien au problème de la prédiction des coûts de projets logiciels. Cependant, la plupart des expérimentations menées, afin de comparer la précision de ses estimations avec celles des autres techniques d'estimation telles que la régression linéaire, les réseaux de neurones, les arbres de décision et les algorithmes génétiques, n'ont pas pu prouver qu'elle performe mieux que les autres dans toutes les situations (Tab. 1.1). En plus, la technique CBR, telle qu'elle a été appliquée en estimation des coûts, utilise encore la logique classique (une proposition ne peut être que *vraie* ou *fausse*) pour formuler une estimation au coût d'un projet logiciel. Par conséquent, elle ne peut traiter convenablement le cas des projets logiciels décrits par des valeurs linguistiques souvent floues et incertaines. Aussi, l'utilisation de la technique CBR en estimation des coûts ne permet pas encore la gestion des incertitudes au niveau des estimations fournies. La section suivante présente des éclaircissements sur la problématique de ces deux limitations.

## 4.3 PROBLÉMATIQUE DE LA GESTION DES IMPRÉCISIONS ET DES INCERTITUDES DANS LE PROCESSUS D'ESTIMATION PAR ANALOGIE

En estimation des coûts, les projets logiciels sont souvent décrits par des valeurs linguistiques plutôt que par des valeurs numériques. Ces valeurs linguistiques sont souvent floues et ambiguës. Cependant, l'application de la technique CBR en estimation des coûts représente et manipule ces valeurs linguistiques par le biais de la logique classique. Ainsi, dans les trois étapes du processus CBR, nous pouvons identifier les anomalies suivantes:

- Dans l'étape de description des projets logiciels, les valeurs linguistiques associées aux différents attributs affectant le coût sont représentées par des intervalles classiques ou des nombres réels.

- Dans l'étape de l'évaluation de la similarité entre les projets logiciels, les mesures utilisées considèrent les attributs mesurés par des valeurs linguistiques comme des variables binaires.

- Dans l'étape d'Adaptation, le choix des projets logiciels les plus similaires au nouveau projet utilise la technique du seuil.

### 4.3.1 Le cas des mesures de similarité

Dans la littérature, la similarité entre deux projets logiciels, décrits par un ensemble d'attributs, est souvent évaluée par la mesure de la distance entre ces deux projets à travers leurs ensembles d'attributs. Ainsi, deux projets logiciels ne sont pas similaires si les différences entre leurs attributs sont flagrantes. Du fait que l'évaluation de la similarité entre deux projets est basée sur la comparaison des deux ensembles d'attributs décrivant respectivement ces deux projets, elle dépend donc de l'environnement étudié. Par conséquent, deux projets logiciels qui sont déclarés similaires dans un environnement peuvent ne pas l'être dans un autre environnement. Selon Fenton et Pfleeger, la similarité est donc un attribut *externe*[7] (Fenton et Pfleeger, 1997). Par conséquent, elle ne peut être mesurée qu'indirectement.

Plusieurs mesures de similarité ont été proposées (Kolodner, 1996). En estimation des coûts, celle utilisant la distance euclidienne dans le cas de projets logiciels décrits par des attributs numériques et la distance d'Égalité dans le cas des attributs qualitatifs, est la plus adoptée pour l'évaluation de la similarité entre les projets logiciels (Shepperd et Schofield, 1997):

$$d(P_1, P_2, V) = \frac{1}{(\sum_{v_j} d_{v_j}(P_1, P_2))^{1/2}} \qquad \text{(Équation 4.1)}$$

où $P_1$ et $P_2$ sont deux projets logiciels, V est l'ensemble des attributs $v_j$ décrivant les deux projets $P_1$ et $P_2$, et

$$d_{v_j}(P_1, P_2) = \begin{cases} (V_j(P_1) - V_j(P_2))^2 & (4.2.1) \\ 1 \text{ si } V_j(P_1) = V_j(P_2) & (4.2.2) \\ 0 \text{ sinon} \end{cases} \qquad \text{(Équation 4.2)}$$

[7] Voir le tableau 3.2 pour la définition d'un attribut Externe.

où 4.2.1) représente la distance euclidienne dans le cas de variables numériques et 4.2.2) représente la distance d'Égalité dans le cas de variables qualitatives.

Shepperd et Schofield ont identifié deux limitations quant à l'utilisation des mesures de similarité en estimation des coûts. Premièrement, elles engagent des calculs très intensif; par conséquent, elles requièrent le développement d'un système CBR pour leur utilisation. Nous citons, en particulier, les deux systèmes ESTOR et ANGEL (*ANaloGy Estimation tooL*) développés respectivement par Vicinanza, Prietula et Mukhopadhyay en 1990 et Shepperd et Schofield en 1997. Deuxièmement, elles manipulent les valeurs linguistiques par la logique classique (deux valeurs binaires 0 et1) (Équation 4.2.2). Ainsi, elles ne considèrent pas les imprécisions souvent attachées aux valeurs linguistiques dans leur processus de mesurage de la similarité.

### 4.3.2 Le cas de l'étape d'Adaptation

Cette étape consiste en l'utilisation des coûts réels des projets logiciels les plus similaires au nouveau projet pour en déduire une estimation à son coût de développement. Pour ce faire, l'estimateur doit choisir:

- Le nombre de projets les plus similaires au nouveau projet, $k$, et
- la formule utilisée pour combiner les coûts réels des projets précédemment choisis afin de déduire une estimation du coût du nouveau projet.

Dans la littérature, il n'y a pas de règles prédéfinies pour le choix du nombre de projets les plus similaires, $k$. En général, le nombre $k$ est fixé à une constante (souvent entre 1 et 10) ou est évalué par le nombre de projets logiciels ayant un degré de similarité, avec le nouveau projet, inférieur ou égal à un seuil. La plupart des expérimentations de validation de ces deux stratégies de choix du nombre $k$ ont avantagé l'utilisation d'une valeur de $k$ inférieure ou égale à 3 (Shepperd et Schofield, 1997; Kadoda et al., 2000; Angelis et Stamelos, 2000).

Cependant, ces deux stratégies de choix du nombre $k$ présentent une limitation critique au niveau de la tolérance des imprécisions dans l'évaluation de la valeur linguistique *les plus*

*similaires*. En effet, supposons par exemple que nous ayons fixé le nombre *k* à 2 et que les trois premiers projets similaires à un nouveau projet P ($P_1$, $P_2$ et $P_3$) aient respectivement les degrés de similarité suivants 3,30, 4,00, et 4,01. Dans ce cas, nous utilisons seulement les deux premiers projets ($P_1$ et $P_2$) pour déduire une estimation du coût du nouveau projet P. Le projet $P_3$ n'est pas considéré malgré la différence minime qui existe entre d(P, $P_2$) = 4,00 et d(P, $P_3$) = 4,01. Nous présumons que l'utilisation du nombre *k* résulte du fait que la valeur linguistique *les plus similaires* est représentée implicitement, dans les deux stratégies du choix du nombre *k*, par la logique classique.

Une autre limitation que nous relevons au niveau de l'étape d'Adaptation est qu'elle fournit une seule valeur numérique du coût estimé. Par conséquent, si cette valeur est largement différente du coût nécessaire pour le développement du logiciel, les gestionnaires pourront commettre des erreurs fatales au niveau de la gestion du projet logiciel. Il est donc préférable que l'étape d'Adaptation génère un ensemble de valeurs possibles, plutôt qu'une seule valeur du coût estimé. Ceci permettra la gestion des incertitudes au niveau des estimations fournies ainsi que de l'évaluation des risques qui leur sont associés. La nécessité de la gestion des incertitudes au niveau des estimations d'un modèle utilisant la technique CBR est justifiée par:

- Une estimation est une évaluation d'une situation future d'une entité particulière (Kitchenham et Linkman, 1997).

- Dans l'étape de description des projets logiciels, on ne considère que les attributs pour lesquels on a des données enregistrées. Ainsi, ceux qui n'ont pas de données historiques ne seront pas pris en considération bien qu'ils puissent être significatifs pour l'estimation des coûts. Cela engendrera donc des incertitudes au niveau du coût estimé.

- Les gestionnaires préfèrent, en général, avoir plusieurs valeurs possibles du coût d'un projet, plutôt qu'une seule. Ceci leur offrira plus de flexibilité au niveau de la gestion des projets logiciels.

- La conséquence de l'affirmation *Similar software projects have similar costs* est imprécise. En effet, cette affirmation confirme seulement que les coûts sont similaires et non pas égaux.

- L'affirmation peut être non-déterministe, c'est-à-dire qu'on peut avoir deux projets logiciels similaires mais leurs coûts sont totalement différents.

Pour remédier à toutes ces limitations relatives à la gestion des imprécisions tout le long du processus CBR et de la gestion des incertitudes au niveau des coûts estimés, nous proposons une version floue de la technique CBR, référée dans la suite par *Fuzzy Analogy*. *Fuzzy Analogy* satisfera ainsi les deux critères fondamentaux de l'intelligence, à savoir la tolérance des imprécisions et la gestion des incertitudes.

## 4.4 DESCRIPTION ET VALIDATION DE LA TECHNIQUE DE GESTION DES IMPRÉCISIONS DANS *FUZZY ANALOGY*

*Fuzzy Analogy* est une «fuzzification» de la procédure classique d'estimation des coûts par analogie. Son objectif est de permettre la tolérance des imprécisions tout au long du processus d'estimation par analogie ainsi que la gestion des incertitudes au niveau des coûts estimés. Son processus d'estimation est composé, comme dans le cas de la procédure classique d'estimation par analogie, de trois étapes: 1) Identification des projets logiciels par un ensemble d'attributs; 2) Évaluation de la similarité entre le nouveau projet et chaque projet logiciel de la base de données; 3) Utilisation des coûts réels des projets logiciels les plus similaires au nouveau projet pour en déduire une estimation à son coût. Chaque étape de notre approche *Fuzzy Analogy* est une «fuzzification» de son équivalence dans la procédure classique d'estimation par analogie. Dans ce qui suit, nous décrivons les principes de chaque étape de notre approche.

### 4.4.1 Identification des projets logiciels

L'objectif de cette étape est la description des projets logiciels par un ensemble d'attributs tels que la fiabilité et la complexité logicielle et la compétence des analystes. La sélection des attributs qui décrivent *mieux* les projets logiciels est une tâche complexe en estimation des coûts par analogie. En effet, les attributs sélectionnés doivent être indépendants et

significatifs pour l'estimation des coûts. Pour le critère de signification, par exemple, la pratique consiste à évaluer la corrélation entre chaque attribut et le coût du projet. Ainsi, si le niveau de cette corrélation est satisfaisant, l'attribut est sélectionné. En plus de l'indépendance et de la signification, il y a deux autres critères qui nous semblent importants. Premièrement, l'attribut sélectionné doit être compréhensible, c'est-à-dire, il doit être bien défini. Deuxièmement, l'attribut sélectionné doit être opérationnel, c'est-à-dire il doit être facilement mesurable.

Shepperd et al., dans leur système ANGEL, ont proposé de résoudre le problème de sélection des attributs par une recherche exploratoire de tous les sous-ensembles possibles d'attributs. Ainsi, ANGEL recherche parmi toutes les combinaisons possibles celle qui a souvent mené à des estimations précises. C'est ce sous-ensemble d'attributs qui sera utilisé pour la description des projets logiciels (Shepperd et Schofield, 1997; Kadoda et al., 2000). Cependant, ils reconnaissent que cette recherche exploratoire est de type NP-Complet; par conséquent, elle n'est possible que dans les situations où le nombre d'attributs candidats est assez petit. En opposé, Briand, Langley et Wieczorek propose l'utilisation de la procédure t-test pour la sélection d'attributs (Briand, Langley et Wieczorek, 2000). Shepperd et al. considèrent que les procédures statistiques telles que le t-test et le coefficient de corrélation sont, en général, insuffisantes pour modéliser les interactions complexes existantes entre les différents attributs d'un projet logiciel (Kadoda et al., 2000).

Bien qu'il soit reconnu que la procédure utilisée, pour la sélection des attributs, influence les performances d'un système CBR, la communauté en estimation des coûts ne lui a pas encore accordé un intérêt particulier. Récemment, Kirsopp, Shepperd et Hart ont étudié l'application de trois techniques heuristiques pour la sélection des attributs dans un système CBR d'estimation des coûts (Kirsopp, Shepperd et Hart, 2002). Cette alternative de recherche nous semble prometteuse car elle étudie l'application des techniques de l'intelligence artificielle au cas des projets logiciels. En effet, le problème de sélection des attributs (*Features Subset Selection* -FSS-) est bel et bien connu en intelligence artificielle (Skalak, 1994; Debuse et Rayward-Smith, 1997; Kohavi et John, 1997). Jusqu'à présent, dans notre système CBR

nommé F_ANGEL[8], nous avons adopté une solution simple au problème FSS. Cette solution consiste à donner le choix à l'estimateur pour sélectionner les attributs qu'il juge pertinents à l'estimation du coût d'un nouveau projet. Cette solution se base donc essentiellement sur l'expertise de l'estimateur. Ainsi, elle est sujette à une grande variation d'un estimateur à un autre, du fait de sa subjectivité. Nous envisageons, dans nos futurs travaux de recherche, d'utiliser des techniques plus appropriées dans F_ANGEL afin d'aider l'estimateur dans le choix du sous-ensemble d'attributs décrivant mieux les projets logiciels.

L'objectif de *Fuzzy Analogy* est de traiter convenablement les imprécisions au niveau de la description des projets logiciels quand certains de leurs attributs sont mesurés sur une échelle composée de valeurs linguistiques. Ainsi, nous représentons ces valeurs linguistiques dans *Fuzzy Analogy* par des ensembles flous plutôt que par des ensembles classiques. Dans le cas des attributs mesurés par des valeurs numériques (aucune imprécision), notre approche garde la même représentation. Supposons l'existence de M attributs, $V_j$, mesurés par des valeurs linguistiques ($A_k^j$). Chaque valeur linguistique, $A_k^j$, est représentée par un ensemble flou ayant une fonction d'appartenance ($m_{A_k^j}$). Il est préférable que les ensembles flous associés aux valeurs linguistiques d'un attribut soient convexes, normaux et forment une partition floue (Sect. 2.4.3). Cette propriété garantit que les ensembles flous se chevauchent entre eux, deux à deux. Par exemple, dans le cas des projets logiciels de la base de données COCOMO'81, nous avons représenté les six valeurs linguistiques (*très bas*, *bas*, moyen, *élevé*, *très élevé*, *extra-élevé*) composant l'échelle du COCOMO'81 par des ensembles flous (Idri, Abran et Kjiri, 2000) (voir Annexe A).

Afin de prendre en considération l'importance de chaque attribut dans le processus d'estimation par *Fuzzy Analogy*, nous affectons des poids de pondération aux attributs sélectionnés, $u_j$. Ces poids $u_j$ dépendent de l'environnement étudié. Par exemple dans le cas des projets de la base de données COCOMO'81, les poids ont été calculés en utilisant le ratio de productivité associé à chaque attribut (Idri et Abran, 2001b). Le tableau 4.1 montre les poids $u_j$ associés aux facteurs du COCOMO'81.

[8] F_ANGEL, *Fuzzy ANaloGy Estimation tooL*, est un système CBR qui implémente notre approche *Fuzzy Analogy*. Nous l'avons développé avec *Visual Basic Microsoft* et *Microsoft Access* (voir Annexe B)

**Tableau 4.1**
Les poids $u_j$ associés aux douze attributs décrivant les projets logiciels COCOMO'81

| Attribut ($V_j$) | Poids $u_j$ | Attribut ($V_j$) | Poids $u_j$ |
|---|---|---|---|
| ACAP | 2.05 | VIRT-MIN | 1.49 |
| PCAP | 2.30 | VEXP | 1.34 |
| TIME | 1.66 | TURN | 1.32 |
| AEXP | 1.57 | DATA | 1.23 |
| STOR | 1.56 | LEXP | 1.20 |
| VIRT-MAJ | 1.49 | SCED | 1.11 |

4.4.2 Évaluation de la similarité entre les projets logiciels

Le choix d'une mesure pour évaluer la similarité entre deux projets logiciels est très important dans le processus d'estimation des coûts par analogie. En effet, les projets sont ordonnés en fonction de leur degré de similarité avec le nouveau projet. Comme nous l'avons mentionné auparavant dans la section 4.3, la plupart des expérimentations utilisaient, pour évaluer la similarité entre les projets logiciels, la distance euclidienne dans le cas des valeurs numériques et la distance d'Égalité dans le cas des valeurs linguistiques (Équations 4.1 et 4.2). En logique floue, une mesure de similarité est une fonction à valeurs dans l'intervalle [0, 1]. Elle doit être réflexive et symétrique (Dubois et al., 1999; Zadeh, 1971). Cependant, le concept de similarité a été aussi débattu dans plusieurs disciplines des sciences cognitives telles que la philosophie, la psychologie et l'intelligence artificielle (Mac Cormac, 1985; Tversky, 1984; Gentner et Markman, 1997; Santini et Jain, 1999; Hüllermeir, 2001).

Du fait que les mesures de similarité utilisées en estimation des coûts ne manipulaient pas convenablement le cas des projets logiciels décrits par des valeurs linguistiques, nous avons proposé un ensemble de mesures qui intègrent dans leurs processus de mesurage des concepts et des outils de la logique floue (Idri et Abran, 2000c). Ces mesures évaluent la similarité globale entre deux projets $P_1$ et $P_2$, $d(P_1, P_2)$, en combinant les similarités individuelles associées aux différents attributs $V_j$ décrivant $P_1$ et $P_2$, $d_{v_j}(P_1, P_2)$ (Fig. 4.2).

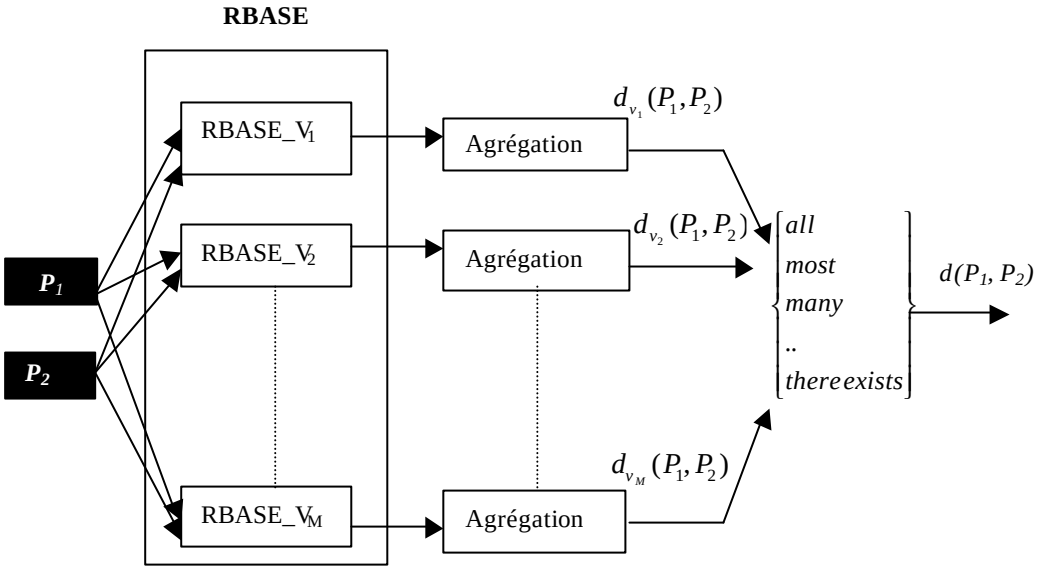



**Figure 4.2** Processus de mesure de la similarité entre deux projets logiciels

Similarités individuelles entre deux projets logiciels $P_1$ et $P_2$

Cette étape consiste à évaluer la similarité entre deux projets $P_1$ et $P_2$ selon chaque attribut $V_j$, $d_{v_j}(P_1, P_2)$. Du fait que chaque attribut $V_j$ est mesuré sur une échelle composée de valeurs linguistiques qui sont représentées par des ensembles flous (Étape d'Identification), $d_{v_j}(P_1, P_2)$ doit exprimer l'égalité floue entre $P_1$ et $P_2$ selon l'attribut $V_j$. Cette égalité floue est représentée par un ensemble flou ayant une fonction d'appartenance à deux variables: $V_j(P_1)$ et $V_j(P_2)$. En logique floue, ce genre d'ensemble flou est nommé une *relation floue*. De telles relations expriment souvent une association ou une corrélation entre les éléments de l'espace produit. Dans notre cas, la relation floue représente l'affirmation suivante: $P_1$ *et* $P_2$ *sont approximativement égaux selon l'attribut* $V_j$. Nous désignons cette relation par $R_{\approx}^{V_j}$. $R_{\approx}^{V_j}$ est une combinaison d'un ensemble de relations floues $R_{\approx,k}^{V_j}$. Chaque $R_{\approx,k}^{V_j}$ représente l'égalité floue selon l'une des valeurs linguistiques $A_k^j$ associée à $V_j$. Ainsi, $R_{\approx,k}^{V_j}$ représente la règle floue Si-Alors dont la prémisse et la conséquence sont deux propositions floues:

$$R_{\approx,k}^{v_j} : if\, V_j(P_1)\ is\ A_k^j\ then\, V_j(P_2)\ is\ A_k^j$$

Par conséquent, pour chaque attribut $V_j$, nous avons une base de règles floues (RBASE_$V_j$) contenant le même nombre de règles que celui de valeurs linguistiques associées à $V_j$. Chaque RBASE_$V_j$ exprime l'égalité floue des deux projets $P_1\ et\ P_2$ selon l'attribut $V_j$, $d_{v_j}(P_1, P_2)$. La base de règle RBASE est composée de toutes les bases RBASE_$V_j$. Elle exprime donc l'égalité floue selon tous les attributs, $d(P_1, P_2)$.

$d_{v_j}(P_1, P_2)$ est déterminée en combinant toutes les règles floues de la base RBASE_$V_j$ afin d'obtenir la relation floue $R_{\approx}^{v_j}$. Donc $d_{v_j}(P_1, P_2)$ n'est en fin de compte que la fonction d'appartenance de la relation floue $R_{\approx}^{v_j}$. La combinaison d'un ensemble de règles floues pour obtenir une seule relation floue est réalisée de différentes façons selon la fonction utilisée pour représenter les implications contenues dans ces règles floues (Sect. 2.8.4). En utilisant les deux cas où une implication floue est représentée par une conjonction ou une implication classique, nous obtenons les trois formules suivantes:

$$d_{v_j}(P_1,P_2) = \begin{cases} \max\limits_k \min\,(\mu_{A_k^j}(P_1),\mu_{A_k^j}(P_2)) & \\ \max - \min\ aggregation & \text{(Équation 4.3)} \\ \sum\limits_k \mu_{A_k^j}(P_1) \times \mu_{A_k^j}(P_2) & \\ sum - product\ aggregation & \text{( Équation 4.4)} \\ \min\limits_k \max\,(1-\mu_{A_k^j}(P_1),\mu_{A_k^j}(P_2)) & \\ \min - Kleene - Dienes\ aggregation & \text{(Équation 4.5)} \end{cases}$$

où $d_{v_j}(P_1, P_2)$=1 signifie que les deux projets $P_1\ et\ P_2$ sont parfaitement similaires selon l'attribut $V_j$, $d_{v_j}(P_1, P_2)$=0 signifie que $P_1\ et\ P_2$ ne sont pas similaires, et $0<d_{v_j}(P_1, P_2)<1$ signifie que $P_1\ et\ P_2$ sont partiellement similaires.

Similarité globale entre deux projets logiciels $P_1$ et $P_2$

La similarité globale entre deux projets $P_1$ et $P_2$, $d(P_1, P_2)$, est évaluée en combinant les similarités individuelles, $d_{v_j}(P_1, P_2)$, par un quantificateur linguistique tel que *all*, *most*, *many*, *at-most a*, et *there exists*. Ce genre de quantificateurs linguistiques est appelé *Regular Increasing Monotone Quantifier* (Yager, 1996). Ceci implique que la similarité globale croît en fonction du nombre de similarités individuelles satisfaites. Par exemple, la similarité globale entre deux projets ayant trois similarités individuelles satisfaites serait plus grande que celle entre ces deux projets s'ils avaient seulement deux similarités individuelles satisfaites. Le choix d'un quantificateur RIM, noté $Q$, dépend des caractéristiques et des spécificités de l'environnement étudié. $Q$ indique la proportion des similarités individuelles nécessaires pour une meilleure évaluation de la similarité globale. Ainsi, la similarité globale entre deux projets $P_1$ et $P_2$ est définie par l'une des expressions informelles suivantes:

$$d(P_1, P_2) = \begin{cases} all\ of\ (d_{v_j}(P_1, P_2)), \\ most\ of\ (d_{v_j}(P_1, P_2)), \\ many\ of\ (d_{v_j}(P_1, P_2)), ou \\ .... \\ there exists\ of\ (d_{v_j}(P_1, P_2)) \end{cases} \qquad \text{(Équation 4.6)}$$

Afin de calculer la similarité globale entre deux projets logiciels, nous devons exprimer formellement l'équation 4.6. Ainsi, nous implémentons le quantificateur RIM choisi (*all*, *most*, ..., *ou there exists of*) de l'équation 4.6 par un opérateur OWA (Ordered Weighted Averaging) (Yager et Kacprzyk, 1997; Yager, 1996). Yager définit un opérateur OWA de dimension M comme une fonction $F: I^M \rightarrow I$ à laquelle on associe un vecteur de poids $W$ de dimension M: $W(w_1, w_2, \ldots, w_M)$ avec :

1- $w_i \in [0,1]$,

2- $\sum_i w_i = 1$, et

3- $F(a_1, .., a_M) = \sum_{j=1}^{M} w_j b_j$ ou $b_j$ est le $j^{ème}$ $a_i$ selon un ordre croissant et les $a_i$ sont les critères à combiner (dans notre cas les $a_i$ sont les similarités individuelles).

.

Le problème est donc de retrouver le vecteur *W* qui sera associé au quantificateur Q que nous adoptons pour combiner les similarités individuelles. Ainsi, la similarité globale entre deux projets $P_1$ et $P_2$ est calculée par:

$$d(P_1, P_2) = \sum_{j=1}^{M} w_j(P_1, P_2)\, d_{v_j}(P_1, P_2) \qquad \text{(Équation 4.7)}$$

où $d_{v_j}(P_1, P_2)$ est la j$^{ème}$ similarité individuelle selon un ordre croissant.. En effet, les similarités individuelles apparaissent dans l'équation 4.7 selon un ordre croissant et non pas selon l'ordre des indices des attributs $V_j$ auxquels elles sont associées. La procédure utilisée pour retrouver le vecteur *W* associé à un quantificateur RIM, *Q*, est composée de deux étapes (Yager, 1996; Idri et Abran, 2001b). Premièrement, le quantificateur *Q* est représenté par un ensemble flou ayant comme univers de discours l'intervalle [0,1]. La fonction d'appartenance associée à *Q* est monotone croissante avec *Q(0)=0* et *Q(1)*= 1. Deuxièmement, les poids $w_j(P_1,P_2)$ sont calculés par:

$$w_j(P_1, P_2) = Q\left(\frac{\sum_{k=1}^{j} u_k}{T}\right) - Q\left(\frac{\sum_{k=1}^{j-1} u_k}{T}\right) \qquad \text{(Équation 4.8)}$$

où $u_k$ est le poids associé au k$^{ème}$ attribut décrivant les projets logiciels (les poids $u_k$ sont fournis dans l'étape d'Identification) et T est la somme des $u_k$. Les poids $w_j(P_1,P_2)$ dépendent de $P_1$ et $P_2$. En effet, le tri des $d_{v_j}(P_1, P_2)$ résulte, dépendamment de $P_1$ et $P_2$, d'un ordre différent des $u_k$. La différence entre la sémantique des poids $u_k$ et celle des poids $w_j$ est que les poids $u_k$ sont utilisés pour pondérer l'importance des attributs dans la description des projets logiciels alors que les poids $w_j$ sont utilisés pour pondérer leurs importances dans l'évaluation de la similarité entre les projets (Idri et Abran, 2001b). Dans le paragraphe suivant, nous illustrons par un exemple notre processus d'évaluation de la similarité globale entre deux projets.

Considérons deux projets $P_1$ et $P_2$ décrits par quatre variables $V_1$, $V_2$, $V_3$ et $V_4$. Les poids $u_k$ associés à ces quatre variables sont respectivement 1, 0,6, 0,5 et 0,9. Le quantificateur RIM choisi est $Q(x)=x^2$. Supposons que l'évaluation des similarités individuelles entre $P_1$ et $P_2$ a

donné respectivement les valeurs suivantes : 0,7, 1, 0,5 et 0,6. Le tri croissant des similarités individuelles donne :

Premièrement : similarité individuelle selon $V_2$, $u_2$=0,6

Deuxièmement : similarité individuelle selon $V_1$, $u_1$=1

Troisièmement : similarité individuelle selon $V_4$, $u_4$=0,9

Quatrièmement : similarité individuelle selon $V_3$, $u_3$=0,5

Nous notons que T=3. Les poids $w_j$ s'obtiennent en appliquant l'équation 4.8 :

$W_1(P_1,P_2)$=Q(0,6/3)-Q(0/3)=0,04; $W_2(P_1,P_2)$= Q(1,6/3)-Q(0,6/3)=0,24; $W_3(P_1,P_2)$= Q(2,5/3)-Q(1,6/3)=0,41; $W_4(P_1, P_2)$= Q(3/3)-Q(2,5/3)=0,31.

Ainsi, la similarité globale entre $P_1$ et $P_2$ est égale à 0,609 (0,609=0,04*1 + 0,24*0,7 + 0,41*0,6 + 0,31*0,5).

Validation axiomatique des mesures de similarité

En métrologie de logiciels, la mesure d'un attribut doit nécessairement représenter les propriétés de cet attribut. Cela nous permettra de choisir parmi un ensemble de mesures candidates pour un attribut, celle qui satisfait le maximum de propriétés de cet attribut. C'est ce qu'on appelle couramment en métrologie de logiciels la validation d'une mesure. Dans la section précédente, nous avons présenté un ensemble de mesures candidates pour l'attribut *similarité entre projets logiciels*. Cette section présente la validation axiomatique de nos mesures de similarités afin de choisir celle(s) qui représente(nt) convenablement les propriétés de l'attribut *similarité entre projets logiciels*.

La validation d'une mesure est l'une des étapes les plus cruciales dans tout le cycle de développement de mesures en génie logiciel. Jusqu'à présent, il n'y a pas une définition commune à ce qu'est une mesure valide. Cependant, plusieurs chercheurs ont adopté différentes approches (mathématiques, empiriques, etc.) pour la validation de mesures en génie logiciel (Zuse, 1994; Kitchenham, Pfleeger et Fenton, 1995; Fenton et Pfleeger, 1997; Jacquet et Abran, 1998; Zuse, 1999). Kitchenham, Pfleeger et Fenton ont discuté ces différentes approches et ont conclu (Kitchenham, Pfleeger et Fenton, 1995):

> «What has been missing so far is a proper discussion of relationship among the different approaches, and how they should be used in practice.»

En l'absence d'une approche unifiée pour la validation d'une mesure en génie logiciel, nous avons choisi de valider nos mesures de similarité entre les projets logiciels en utilisant l'approche de Fenton et Pfleeger (Fenton et Pfleeger, 1997):

> «Validating a software measure is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied.»

Ainsi, nos mesures de similarité, $d_{v_j}(P_1, P_2)$ et $d(P_1, P_2)$, satisfont la condition de représentation si elles ne contredisent aucune notion intuitive préétablie empiriquement sur la similarité entre deux projets logiciels $P_1$ et $P_2$. Ces notions intuitives sont codifiées par des axiomes. Nous avons identifié quatre axiomes qui représentent notre intuition sur la similarité entre les projets logiciels. Le Tableau 4.2 résume les résultats obtenus de cette validation axiomatique de nos mesures de similarité (Idri et Abran, 2001a).

**Axiome 1** (Cet axiome concerne seulement les mesures individuelles, $d_{v_j}(P, P_i)$):

*La similarité entre deux projets logiciels selon un attribut $V_j$, n'est pas nulle si et seulement si les deux projets ont des degrés d'appartenance différents de zéro pour au moins un même ensemble flou de $V_j$*

$$d_{v_j}(P,P_i) \neq 0 \;\; ssi \; \exists \, A_k^j \; tel \; que \; m_{A_k^j}(P) \neq 0 \; and \; m_{A_k^j}(P_i) \neq 0$$

**Axiome 2**

*Une mesure de similarité S est toujours positive:*

$$S(P_1, P_2) \geq 0; \; S(P, P) > 0$$

**Axiome 3**

*Le degré de similarité entre un projet P et lui-même doit être supérieur ou égal à celui de P avec tout autre projet $P_i$:*

$$S(P, P_i) \leq S(P, P)$$

**Axiome 4**

*Une mesure de similarité S est symétrique:*

$$S(P_1, P_2) = S(P_2, P_1)$$

En analysant les résultats de cette validation axiomatique, nous remarquons que $d_{v_j}(P, P_i)$, utilisant l'agrégation *max-min*, satisfait tous les quatre axiomes. Donc, selon Fenton et Pfleeger, elle est une mesure valide. $d_{v_j}(P, P_i)$ utilisant l'agrégation *sum-product* ne satisfait pas l'Axiome 3; bien que cet axiome soit important, nous l'avons retenue et cela pour trois raisons principales (Pour les détails voir (Idri et Abran, 2001a)):

- La différence entre $d_{v_j}(P, P_i)$ et $d_{v_j}(P, P)$ est négligeable quand les ensembles flous associés à une variable $V_j$ satisfont NC[3]. Spécifiquement, cette différence est dans l'intervalle [-1/8,0] quand l'Axiome 3 n'est pas satisfait.
- L'agrégation *sum-product* satisfait les trois autres axiomes.
- Selon Zuse, la validation axiomatique n'est pas toujours nécessaire et souvent, en pratique, on se restreint à montrer que la mesure proposée est une composante d'un système de prédiction valide (Zuse, 1998).

**Tableau 4.2**

Résultats de la validation axiomatique des mesures $d_{v_j}(P, P_i)$ et $d(P, P_i)$

| Axiomes | $d_{v_j}(P, P_i)/d(P, P_i)$ | | |
|---|---|---|---|
| | *max-min* | *sum-product* | *min-Kleene-Dienes* |
| Axiome 1 | Oui/ | Oui/ | Non/ |
| Axiome 2 | Oui/Oui | Oui/Oui | Oui/Oui |
| Axiome 3 | Oui/Oui | Non/Non | Oui /Oui si NC |
| Axiome 4 | Oui/Oui | Oui/Oui | Non/Non |

[3] NC est l'abréviation de *Normal Condition*. Un tuplet ($A_1$, $A_2$, .., $A_n$) d'ensembles flous satisfait NC si tous les $A_i$ sont convexes, normaux, et forment une partition floue. Ici, la mesure min-Kleene-Dienes satisfait l'Axiome 3 si les ensembles flous associés à chaque attribut, sélectionné dans l'étape d'Identification, respectent NC.

### 4.4.3 Adaptation

L'objectif de cette étape est de déduire une estimation du coût du nouveau projet P, en utilisant les coûts réels des projets logiciels les plus similaires à P. Dans la section 4.3, nous avons relevé deux questions importantes relatives aux procédures classiques utilisées dans l'étape d'Adaptation en estimation des coûts:

- Quel est le nombre de projets les plus similaires à P qui sont considérés pour déduire une estimation du coût du projet P?
- Quelle est la formule utilisée pour combiner les coûts réels des projets les plus similaires à P?

Nous présentons ci-dessous les deux solutions adoptées par *Fuzzy Analogy* afin de pallier aux inconvénients des approches classiques.

Nombre de projets

Dans notre approche *Fuzzy Analogy*, nous proposons une nouvelle stratégie pour sélectionner les projets à prendre en considération dans l'étape d'Adaptation. Cette stratégie se base sur les degrés de similarités, $d_{v_j}(P,P_i)$, et la définition adoptée à la proposition $P_i$ *est étroitement similaire à* $P$ dans l'environnement concerné. Intuitivement, un projet $P_i$ est étroitement similaire à P si son degré de similarité à $P$ est approximativement égal à un. Ainsi, une manière adéquate de réaliser cette intuition est de représenter la qualification *étroitement similaire* par un ensemble flou défini dans l'intervalle [0,1]. En effet, cette qualification est imprécise et dépend des caractéristiques de l'environnement étudié. Par conséquent, différentes organisations peuvent adopter différentes définitions de cette qualification. La figure 4.3 montre un exemple d'une représentation possible de la qualification *étroitement similaire*. Dans cet exemple, tout projet logiciel ayant un degré de similarité avec $P$ supérieur ou égal à 0,5, contribuera à l'évaluation d'une estimation au coût de $P$; la contribution de chaque projet $P_i$ est pondérée par $m_{voisinage\ de\ 1}(d(P,P_i))$. Donc $m_{voisinage\ de\ 1}(d(P,P_i))$ représente le degré de vérité de la proposition $P_i$ *est étroitement similaire à* $P$.

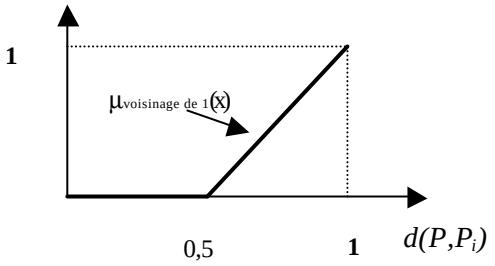



**Figure 4.3** Exemple d'une représentation de la qualification *étroitement similaire* par un ensemble flou.

Formule d'Adaptation

Dans la littérature, la formule d'Adaptation la plus utilisée pour déduire une estimation du coût d'un nouveau projet P est celle utilisant la moyenne arithmétique (pondérée) ou la médiane des k premiers projets les plus similaires à P. Dans le cas d'une moyenne arithmétique pondérée, les poids sont les degrés de similarité ou les rangs des projets selon leurs degrés de similarité à P. Dans l'étape d'Adaptation de *Fuzzy Analogy*, nous utilisons la moyenne arithmétique pondérée de tous les projets historiques. Les poids sont les degrés d'appartenance à l'ensemble flou représentant la qualification *étroitement similaire*. La formule d'Adaptation utilisée est:

$$Effort(P) = \frac{\sum_{i=1}^{N} m_{voisinage\,de\,1}(d(P,P_i)) \times Effort(P_i)}{\sum_{i=1}^{N} m_{voisinage\,de1}(d(P,P_i))} \qquad \text{(Équation 4.9)}$$

Il y a deux avantages à l'utilisation de cette formule par rapport à celles utilisées dans la littérature:

- Elle généralise la formule classique de la moyenne arithmétiques des k premiers projets. En effet, il suffit de choisir $m_{voisinage\,de\,1}(x) = x$ dans l'équation 4.9, c'est-à-dire que la fonction d'appartenance de l'ensemble flou représentant la valeur linguistique *étroitement similaire* est l'Identité, pour retrouver la formule classique.

- Elle est facilement adaptable aux besoins et aux spécificités de chaque environnement. En effet, il suffit de redéfinir la qualification *étroitement similaire* de telle façon qu'elle soit bien adaptée à l'environnement concerné.

### 4.4.4 Validation empirique de *Fuzzy Analogy*

La validation empirique d'un modèle d'estimation des coûts consiste en l'évaluation de la précision de ses estimations de coûts sur une base de projets logiciels. Dans notre cas, nous utilisons les projets logiciels de COCOMO'81 pour évaluer la précision des estimations de notre approche *Fuzzy Analogy*. Cette évaluation utilise le prototype logiciel F_ANGEL qui implémente notre approche. Les résultats de cette validation empirique sont comparés à ceux de trois autres techniques d'estimation: Estimation par analogie classique, COCOMO'81 Intermédiaire, et *fuzzy* COCOMO'81 Intermédiaire (Chap. 3). Les indicateurs utilisés pour l'appréciation de la précision des estimations sont la MRE, la MMRE et le Pred(p) (Set. 1.7).

La base de projets COCOMO'81 est composée de 63 projets logiciels qui sont décrits par 17 attributs: la taille du logiciel mesurée en terme de KISL, le mode du projet (organique, semi-détaché ou intégré) et 15 autres facteurs représentant l'environnement de développement tels que l'expérience des analystes, la fiabilité logicielle et la volatilité de la machine virtuelle. Ces 15 facteurs sont mesurés sur une échelle composée de six valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé*, *très élevé*, et *extra-élevé*. Parmi, ces 15 facteurs, 12 ont été représentés par des ensembles flous (voir Annexe A**).** Ce sont donc ces 12 facteurs qui seront utilisés pour la description des 63 projets logiciels du COCOMO'81. Nous adoptons les poids associés à ces 12 facteurs dans le tableau 4.1. Les quantificateurs linguistiques RIM utilisés, *Q*, dans l'évaluation de la similarité globale entre deux projets logiciels sont définis par l'équation 4.10 du fait que la base COCOMO'81 ne contient pas les informations nécessaires pour déterminer le quantificateur adéquat. En plus, ce genre de quantificateurs de l'équation 4.10 peuvent se rapproher avec une grande précision de plusieurs types de quantificateurs RIM. Enfin, nous utilisons les mêmes trois bases de projets déduites de la base de données originale du COCOMO'81 pour évaluer la précision des estimations de *Fuzzy Analogy* (Voir Chap. 3).

$$Q(r) = r^a \quad a > 0 \qquad \text{(Équation 4.10)}$$

L'analyse des résultats de l'évaluation de la précision des estimations de *Fuzzy Analogy* montre que celle-ci dépend du quantificateur linguistique utilisé pour l'évaluation de la similarité globale ($\alpha$) (Tab. 4.3). En général, nous constatons que la précision des estimations, mesurée en terme de Pred(20), est monotone croissante en fonction de $\alpha$. Cela résulte du fait que nos mesures de similarité sont monotones décroissantes en fonction de $\alpha$:

- Quand $\alpha$ tend vers zéro, l'évaluation de la similarité globale ne prend en considération que peu d'attributs de ceux utilisés dans la description des projets logiciels; le minimum est de considérer un seul attribut. C'est le cas du quantificateur linguistique *there exists* (ligne *max* du tableau 4.3) qui ne considère que l'attribut dont la similarité individuelle est la plus élevée. Ainsi, il est fort probable de trouver, pour deux projets logiciels quelconques de la base de données COCOMO'81, un attribut dont la valeur linguistique associée est la même pour les deux projets. Par conséquent, leur similarité globale serait élevée (au voisinage de 1).

- Quand $\alpha$ tend vers l'infini, l'évaluation de la similarité globale prend en considération la plupart des attributs décrivant les projets logiciels; le maximum doit considérer tous les attributs. C'est le cas du quantificateur linguistique *all* (ligne *min* du tableau 4.3). Ainsi, il est fort probable de trouver, pour deux projets logiciels quelconques de la base de données COCOMO'81, un attribut dont les valeurs linguistiques associées à ces deux projets sont différentes. Par conséquent, leur similarité globale serait faible (au voisinage de 0).

En plus, nous avons comparé les résultats de la validation de notre approche *Fuzzy Analogy* avec trois autres techniques d'estimation: Estimation par analogie classique, COCOMO'81 Intermédiaire, et *fuzzy* COCOMO'81 Intermédiaire (Idri, Abran, Robert et Khoshgoftaar, 2002b):

- La précision des estimations de *Fuzzy Analogy*, quand $\alpha$ tend vers l'infini, est meilleure que celle de la procédure classique d'estimation par analogie. Dans cette procédure classique, nous avons utilisé la distance euclidienne et la distance d'Égalité pour évaluer la similarité entre les projets logiciels; tous les attributs ont été considérés dans cette évaluation. La meilleure précision des estimations est obtenue

quand nous considérons seulement les deux premiers projets dans l'étape d'Adaptation (k=2 dans la dernière colonne du tableau 4.4 ).

- La précision du COCOMO'81 Intermédiaire est meilleure que celle de l'Analogie classique. Cependant, quand nous intégrons la logique floue dans l'Analogie classique, *Fuzzy Analogy* génère des estimations plus précises que celles du COCOMO'81 Intermédiaire.

Notre validation empirique des quatre techniques d'estimation, sur la base de projets du COCOMO'81, nous a permis de les classifier comme suit, selon deux critères: la précision des estimations et l'adéquation de traitement des imprécisions (Idri, Abran Robert et Khoshgoftaar, 2002b): 1) *Fuzzy Analogy*, 2) *fuzzy* COCOMO'81 Intermédiaire, 3) COCOMO'81 Intermédiaire, et 4) Analogie classique.

**Tableau 4.3**

Résultats de la validation de *Fuzzy Analogy* sur les trois bases de projets logiciels

| | Base de projets | | | | | |
|---|---|---|---|---|---|---|
| | $DB_1$ | | $DB_2$ | | $DB_3$ | |
| a -**RIM** | Pred(0.20) (%0) | MMRE (%) | Pred(0.20) (%) | MMRE (%) | Pred(0.20) (%) | MMRE (%) |
| **Max (there exists)** | 4,76 | 1801,48 | 4,76 | 2902,49 | 4,76 | 1789,64 |
| **1/100** | 4,76 | 1798,85 | 4,76 | 2894,28 | 4,76 | 1786,20 |
| **1/30** | 4,76 | 1792,70 | 4,76 | 2875,26 | 4,76 | 1778,21 |
| **1/15** | 4,76 | 1783,91 | 4,76 | 2848,44 | 4,76 | 1766,86 |
| **1/10** | 4,76 | 1757,13 | 4,76 | 2822,64 | 4,76 | 1755,64 |
| **1/7** | 4,76 | 1763,86 | 4,76 | 2788,70 | 4,76 | 1741,29 |
| **1/3** | 6,34 | 1714,20 | 6,34 | 2648,90 | 6,34 | 1679,67 |
| **1** | 6,34 | 1550,89 | 3,17 | 2258,36 | 7,93 | 1491,93 |
| **3** | 6,34 | 1168,24 | 9,52 | 1571,48 | 9,52 | 1102,40 |
| **7** | 9,52 | 633,99 | 14,28 | 830,79 | 9,52 | 603,76 |
| **10** | 15,87 | 371,84 | 20,63 | 525,45 | 23,80 | 385,90 |
| **15** | 38,09 | 143,92 | 36,50 | 284,20 | 44,44 | 206,19 |
| **30** | 74,60 | 20,40 | 77,77 | 160,38 | 66,66 | 62,83 |
| **100** | 92,06 | 4,06 | 84,12 | 30,37 | 87,63 | 23,24 |
| **Min (all)** | 92,06 | 4,03 | 87,30 | 29,12 | 88,88 | 21,99 |

**Tableau 4.4**
Résultats de la validation du COCOMO'81 Intermédiaire, *fuzzy* COCOMO'81 Intermédiaire et l'Analogie classique

| | *fuzzy*/original COCOMO'81 Intermédiaire | | | | | | Analogie classique (trois bases) | |
|---|---|---|---|---|---|---|---|---|
| | $DB_1$ | | $DB_2$ | | $DB_3$ | | K | *Pred(0.20)*% |
| *Pred(20)*( %) | 62.14 | 68 | 46.86 | 68 | 41.27 | 68 | **2** | 31.75 |
| Min MRE(%) | 0.11 | 0.02 | 0.40 | 0.02 | 0.06 | 0.02 | **3** | 25.40 |
| Max MRE(%) | 88.60 | 83.58 | 3233.03 | 83.58 | 88.03 | 83.58 | **4** | 19.05 |
| Moyenne $MRE_i$(%) | 22.50 | 18.52 | 78.45 | 18.52 | 30.80 | 18.52 | **5** | 12.70 |
| DS MRE | 19.69 | 16.97 | 404.40 | 16.97 | 22.95 | 16.97 | **6** | 12.70 |

## 4.5 GESTION DES INCERTITUDES AU NIVEAU DES ESTIMATIONS DE *FUZZY ANALOGY*

La gestion des incertitudes au niveau des estimations de coûts fournies par *Fuzzy Analogy* signifie qu'elle doit générer un ensemble de valeurs estimées, plutôt qu'une seule, au coût d'un projet avec une distribution de possibilités. Cette fonction de distribution indique les degrés de certitude associés aux différentes valeurs possibles du coût d'un projet logiciel. Kitchenham et Linkman ont examiné quatre sources d'incertitude dans un modèle d'estimation des coûts: les erreurs de mesurage des attributs affectant le coût, les erreurs du modèle, les erreurs relatives aux hypothèses faites sur le modèle et les erreurs relatives aux caractéristiques de l'environnement à partir duquel le modèle a été mis au point (Kitchenham et Linkman, 1997). Les deux premières sources d'incertitudes sont dépendantes respectivement de la précision des mesures des attributs décrivant le projet logiciel et de la précision des estimations du modèle. Les deux autres sources d'incertitude sont attachées aux hypothèses faites sur les entrées inconnues du modèle au moment de la formulation d'une estimation au coût d'un projet. Dans cette section, nous étudions seulement les deux premières sources d'incertitude vu que les deux autres sont spécifiques à la gestion des risques associés aux incertitudes des estimations d'un modèle (Kithcenham et Linkman, 1997).

### 4.5.1 Incertitudes relatives aux erreurs dans le mesurage des attributs décrivant les projets logiciels

Notre approche *Fuzzy Analogy* s'intéresse davantage aux cas des projets logiciels décrits par des valeurs linguistiques plutôt que par des valeurs numériques. L'utilisation des valeurs linguistiques, en plus des valeurs numériques, dans un processus de mesurage d'un attribut logiciel, a plusieurs avantages (Zadeh, 2001; Idri, Abran, Robert et Khoshgoftaar, 2002b):

- Elles sont faciles à comprendre contrairement au cas des valeurs numériques.
- Elles permettent la tolérance des imprécisions dans le processus de mesurage.
- Elles généralisent les valeurs numériques qui ne sont utilisées que dans le cas de disponibilité d'informations précises sur un projet logiciel (imprécision est égale à zéro). Malheureusement, souvent ce n'est pas le cas en génie logiciel.
- Elles permettent d'exprimer convenablement les capacités limitées de l'esprit humain dans le traitement des informations précises et infinies.

Dans notre cas, l'utilisation des valeurs linguistiques dans la description des projets logiciels permet de réduire les effets des erreurs commises, au niveau du mesurage des attributs d'un projet logiciel, sur les estimations fournies par *Fuzzy Analogy*. En effet, dans le cas de valeurs numériques, les erreurs de mesurage des attributs sont évaluées par la différence entre les valeurs réelles et les valeurs générées par les mesures. Par contraste, dans le cas des valeurs linguistiques, ces erreurs sont évaluées par la différence entre leurs degrés d'appartenance relatifs aux valeurs linguistiques. Ainsi, les effets des erreurs de mesurage dans le cas des valeurs numériques peuvent être plus influentes sur la précision des estimations du modèle qu'ils le sont dans le cas des valeurs linguistiques. Par exemple, considérons le cas de l'attribut *expérience des programmeurs* mesuré en termes de nombre d'années d'expérience. Aussi, supposons que nous pouvons associer à chaque programmeur une des trois qualifications (*bas*, *moyen* et *élevé*) dépendamment de son expérience (Fig. 4.4).

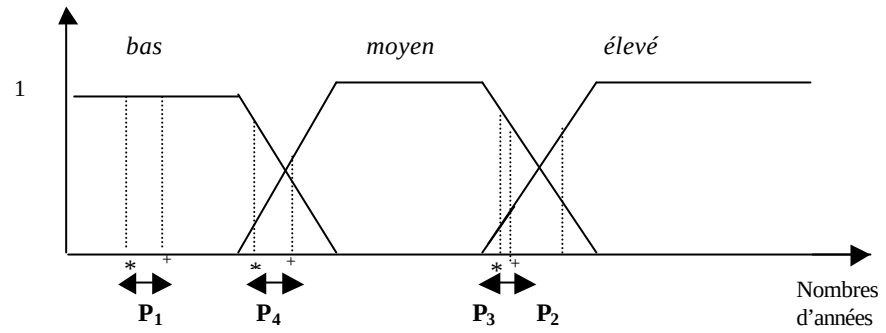



**Figure 4.4** Exemples de situations où les erreurs de mesurage des attributs affectent (ou non) les estimations fournies par *Fuzzy Analogy*.

La figure 4.4 illustre plusieurs situations où les erreurs de mesurage de l'attribut *expérience des programmeurs* n'ont aucun effet sur les estimations des coûts générées par *Fuzzy Analogy*:

- $P_1$ est à la position (*) au lieu de la position (+): dans les deux cas $P_1$ appartient à la valeur linguistique *bas* avec un degré d'appartenance égal à un. Par conséquent, cette erreur de mesurage, commise sur l'expérience du programmeur $P_1$, n'affecte pas l'estimation du coût. Cependant, si nous utilisons le nombre d'années d'expérience pour évaluer l'attribut *expérience de* $P_1$, l'erreur de mesurage pourra affecter l'estimation du coût.

- $P_3$ est à la position (*) au lieu de la position (+): $P_3$ appartient aux deux valeurs linguistiques *moyen* et *élevé* avec deux degrés d'appartenance différents. Si nous évaluons la similarité individuelle de $P_3$ et $P_2$ par l'agrégation *max-min*, nous obtenons la même valeur dans les deux cas ($P_3$ est dans la position (*) ou (+)).

L'utilisation de quantificateurs linguistiques RIM pour l'évaluation de la similarité globale entre deux projets permet aussi, dans certaines situations, d'éviter les effets des erreurs de mesurage des attributs sur l'estimation du coût d'un projet logiciel. Par exemple, considérons que nous avons choisi le quantificateur linguistique *au moins quatre* pour combiner les différentes similarités individuelles, et que tous les attributs décrivant les projets logiciels ont les mêmes poids $u_k$. Dans ce cas, la similarité globale de ces deux projets est exactement

égale à la quatrième similarité individuelle selon un ordre croissant de ces deux projets. Ainsi, si l'attribut sur lequel des erreurs de mesurage ont été commises a une similarité individuelle classée après la quatrième position dans un ordre croissant, ces erreurs n'auront aucun effet sur l'estimation du coût.

Bien que ces exemples illustrent des situations où les erreurs de mesurage n'ont pas d'effet sur les estimations générées par *Fuzzy Analogy*, nous admettons que dans certains cas, elles peuvent influencer l'estimation du coût. Par exemple, dans la figure 4.4, la similarité individuelle entre $P_1$ et $P_4$ ($P_4$ est dans la position (*)) est différente de celle entre $P_1$ et $P_4$ ($P_4$ est dans la position (+)). Ainsi, si ni le quantificateur linguistique utilisé dans l'évaluation de la similarité globale et ni la définition adoptée pour la valeur linguistique *étroitement similaire* ne permettent de masquer cette différence, elle aura des effets d'imprécision et d'incertitude sur l'estimation du coût fournie par *Fuzzy Analogy.*

En conclusion, les erreurs de mesurage sur un attribut peuvent être masquées au cours des trois étapes de *Fuzzy Analogy.* Ainsi, nous nous sommes intéressés plus à la deuxième source d'incertitude, à savoir celle associée à la précision des estimations de notre approche *Fuzzy Analogy.*

### 4.5.2 Incertitudes relatives aux imprécisions dans les estimations fournies par *Fuzzy Analogy*

Ce type d'incertitude est dû aux imprécisions des estimations générées par *Fuzzy Analogy*. En général, la précision d'un modèle d'estimation des coûts dépend de trois paramètres:

- L'approche adoptée pour la mise au point du modèle (régression linéaire, réseaux de neurones, CBR, etc.),
- les caractéristiques de la base de projets logiciels à partir duquel le modèle a été mis au point, et
- les caractéristiques de la base de projets logiciels qui servira pour la validation du modèle.

La précision des estimations d'un modèle est souvent évaluée par l'indicateur MMRE (Moyenne des Erreurs Relatives -MRE-). Ainsi, si l'effort estimé d'un projet par un modèle, ayant une MMRE égale à 25%, est de 300Homme-jours, il est fort probable que la valeur réelle de l'effort de ce projet est dans l'intervalle [225, 375]. L'évaluation de la précision du *Fuzzy Analogy* sur la base de projets COCOMO'81 a donné une MMRE égale à 21% dans le cas du quantificateur linguistique *all*. Par conséquent, pour chaque estimation du *Fuzzy Analogy*, nous avons une incertitude de ±21%. Cette incertitude pourra être, par exemple, représentée par la fonction de distribution, *F*, de la figure 4.5. *F(x)* représente la probabilité pour que la valeur réelle de l'effort soit égale à *x*.

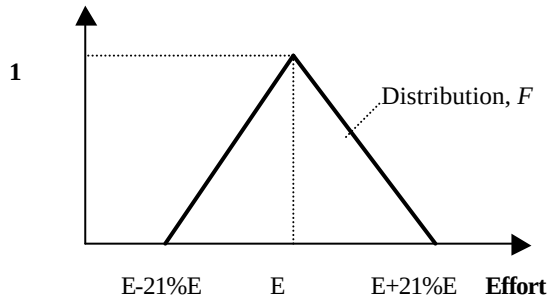



**Figure 4.5** Exemple d'une distribution pour la gestion des incertitudes d'une estimation E.

Kitchenham et Linkman critiquent l'utilisation du MMRE pour la gestion des incertitudes par le fait qu'elle accorde les mêmes degrés d'incertitude aux sur-estimations aussi bien qu'aux sous-estimations. Ils proposent l'utilisation d'une distribution de probabilité telle que la distribution Gamma pour la gestion de ces incertitudes (Kitchenham et Linkman, 1997). Nous nous sommes basés sur leur approche pour la gestion des incertitudes au niveau des estimations du *Fuzzy Analogy*. Cependant, la distribution de probabilité que nous adoptons dépendra des caractéristiques de l'environnement étudié. Cette distribution sera déterminée en utilisant la théorie de possibilité de Zadeh (Zadeh, 1971).

*Fuzzy Analogy* est basée sur l'affirmation suivante *Similar software projects have similar costs*. Cette affirmation comprend deux sources d'incertitude: premièrement, sa conséquence est imprécise; deuxièmement, elle peut être non-déterministe. Dubois et al. ont proposé deux approches différentes pour la reformulation de cette affirmation selon qu'elle est déterministe ou non-déterministe (Dubois et al., 1999). Ainsi, la question principale que nous devons

examiner est: l'affirmation *Similar software projects have similar costs* est-elle déterministe ou non-déterministe en estimation des coûts de développement de logiciels?

Intuitivement, il n'y a aucune raison pour que deux projets similaires n'aient pas des coûts similaires. Cependant, dans la pratique, nous pouvons citer deux sources, souvent omniprésentes, qui mènent à l'opposé de cette intuition:

1- Les projets logiciels ne peuvent être décrits d'une manière précise et exacte, et

2- les mesures de similarité utilisées entre les projets logiciels (*d*) ainsi que celles utilisées entre leurs coûts (*C*) peuvent être non valides.

Au niveau empirique, nous avons mené une expérimentation sur la base de projets logiciels du COCOMO'81 pour évaluer la nature de l'affirmation *Similar software projects have similar costs* (déterministe ou non-déterministe). Ainsi, nous évaluons la similarité entre chaque couple de projets $(P_i, P_j)$ du COCOMO'81 et nous comparons $d(P_i, P_j)$ avec $C(c_i, c_j)$, $c_i$ et $c_j$ sont respectivement les coûts réels de $P_i$ et $P_j$:

- Si $d(P_i, P_j) \leq C(c_i, c_j)$, l'affirmation est déterministe pour le couple $(P_i, P_j)$. Cela signifie que la valeur de vérité de la proposition $P_i$ *et* $P_j$ *sont similaires* doit être inférieure ou égale à celle de la proposition $c_i$ *et* $c_j$ *sont similaires*.

- Si $d(P_i, P_j) > C(c_i, c_j)$, l'affirmation est non-déterministe pour le couple $(P_i, P_j)$.

Pour l'évaluation de la similarité individuelle, nous avons utilisé l'agrégation *max-min* du fait qu'elle satisfait nos quatre axiomes (Tab. 4.2). Les quantificateurs linguistiques utilisés pour l'évaluation de la similarité globale sont ceux définis par l'équation 4.10.

Le coût de développement d'un projet logiciel est une valeur numérique. Ainsi, l'ensemble des valeurs similaires au coût d'un projet logiciel dépend, en général, des spécificités et des contraintes de l'environnement impliqué. Nous présentons ci-dessous deux mesures de similarité entre les coûts de logiciels (Fig. 4.6):

1- La mesure C_R qui définit l'ensemble des valeurs similaires au coût $C_0$ par l'intervalle $[C_0-pC_0, C_0+pC_0]$, p est un pourcentage à choisir.

2- La mesure C_A qui définit l'ensemble des valeurs similaires au coût $C_0$ par l'intervalle $[C_0-cst, C_0+cst]$, cst est constante à choisir.

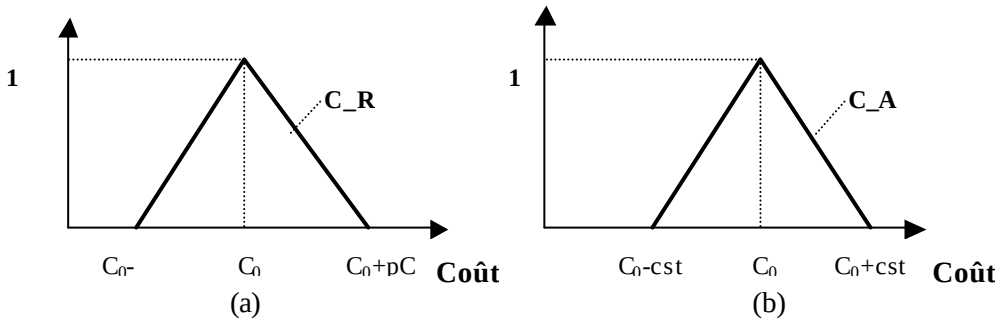



**Figure 4.6** Deux exemples de mesures de similarité entre les coûts de logiciels: (a) C_R, (b) C_A.

La mesure C_R , par contraste à la mesure C_A, prend en considération la valeur $C_0$ dans la définition de l'intervalle des valeurs qui lui sont similaires. Par conséquent, la longueur de cet intervalle dépend de $C_0$. Ainsi, la mesure C_R n'est pas symétrique. Le débat sur la nécessité de la propriété de symétrie pour une mesure de similarité dépasse le cadre de cette thèse. En effet, selon Dubois et al., la symétrie est une propriété nécessaire que n'importe quelle mesure de similarité doit satisfaire (Dubois et al., 1999); par contraste, Tversky suggère que sa nécessité dépend du contexte (Tversky, 1984). Dans notre cas par exemple, une mesure de similarité entre les projets logiciels doit être symétrique; à l'opposé, une mesure de similarité entre leurs coûts peut être non-symétrique. En effet, pour des raisons de *business*, nous pouvons considérer, par exemple, que 15 K$ est plus similaire à 20 K$ que 20 K$ l'est à 15 K$. La mesure C_R répond bien à cette réflexion et elle nous semble être très utile en estimation du coût. D'ailleurs, nous pouvons constater qu'elle reflète bien le principe couramment utilisé dans la validation des modèles d'estimation des coûts. En effet, l'évaluation de la précision des estimations d'un modèle est basée sur l'affirmation: *Une estimation est acceptable si son MRE est inférieure ou égale à 25*. Cette affirmation est équivalente à: *Une estimation est acceptable si elle est similaire, selon la mesure C_R, à la valeur réelle du coût*, avec un pourcentage p est égal à 25.

**Tableau 4.5**
Résultats de l'évaluation de l'affirmation *Similar software projects have similar costs* sur la base des projets du COCOMO'81

| | C_A: cts=6 *(6 est la valeur minimale des coûts des projets de la base COCOMO'81)* | | | | C_R: p=25 | | | |
|---|---|---|---|---|---|---|---|---|
| a -**RIM** | NB_CBR_ND | Min $E_{ij}$ | Max $E_{ij}$ | Moyenne $E_{ij}$ | NB_CBR_ND | Min $E_{ij}$ | Max $E_{ij}$ | Moyenne $E_{ij}$ |
| **Max** | 3899 | 0,1666 | 1 | 0,9825 | 3899 | 0,0446 | 1 | 0,9652 |
| **1/10** | 3899 | 0,0839 | 0,9956 | 0,9088 | 3897 | 0,0083 | 0,9924 | 0,9133 |
| **1/7** | 3899 | 0,0507 | 0,9937 | 0,8813 | 3889 | 0,0039 | 0,9893 | 0,8938 |
| **1/5** | 3899 | 0,0826 | 0,9913 | 0,8813 | 3887 | 0,0062 | 0,9851 | 0,8668 |
| **1/3** | 3887 | 0,0003 | 0,9858 | 0,8239 | 3875 | 0,0043 | 0,9755 | 0,8092 |
| **1** | 3857 | 0,0041 | 0,9606 | 0,5953 | 3799 | 0,0085 | 0,9313 | 0,5887 |
| **10** | 3769 | 2,5E-08 | 0,8109 | 0,8109 | 3634 | 2,5E-08 | 0,6526 | 0,0401 |
| **20** | 3767 | 6,2E-16 | 0,7633 | 0,0120 | 3625 | 6,2E-16 | 0,5889 | 0,0114 |
| **100** | 3765 | 9,4E-77 | 0,7276 | 0,0047 | 3623 | 9,4E-77 | 0,5769 | 0,0042 |
| **500** | 3755 | 8,3E-229 | 0,7276 | 0,0047 | 3613 | 2,6E-305 | 0,5769 | 0,0042 |
| **1000** | 3487 | 5,3E-310 | 0,7276 | 0,0050 | 3355 | 5,3E-310 | 0,5769 | 0,0045 |
| **2000** | 2155 | 2,2E-310 | 0,7276 | 0,0082 | 2080 | 2,2E-310 | 0,5769 | 0,0073 |
| **3000** | 1263 | 1,0E-307 | 0,7276 | 0,0140 | 1219 | 1,0E-307 | 0,5769 | 0,0125 |
| **4000** | 977 | 2,6E-309 | 0,7276 | 0,0181 | 930 | 2,6E-309 | 0,5769 | 0,0164 |
| **5000** | 433 | 3,9E-266 | 0,7276 | 0,0408 | 415 | 3,9E-266 | 0,5769 | 0,0367 |
| **8000** | 401 | 1,6E-299 | 0,7276 | 0,0740 | 229 | 1,6E-299 | 0,5769 | 0,0666 |
| **10000** | 239 | 3,7E-310 | 0,7276 | 0,1022 | 166 | 3,7E-310 | 0,5769 | 0,0918 |
| **20000** | 77 | 0,0121 | 0,7276 | 0,2297 | 72 | 0,0121 | 0,5769 | 0,2118 |
| **Min** | 77 | 0,0121 | 0,7276 | 0,2297 | 72 | 0,0121 | 0,5769 | 0,2118 |

Le tableau 4.5 résume les résultats de l'évaluation de l'affirmation *Similar software projects have similar costs* sur la base de projets du COCOMO'81. Dans cette expérimentation, nous avons utilisé plusieurs quantificateurs linguistiques pour l'évaluation de la similarité entre les projets logiciels (colonne $\alpha$-RIM). Pour l'évaluation de la similarité entre leurs coûts, nous avons utilisé les deux mesures C_A et C_R. Pour tous les projets du COCOMO'81, nous calculons les quantités suivantes:

- NB_CBR_ND: le nombre de cas des couples $(P_i,P_j)$ où l'affirmation *Similar software projects have similar costs* est non-déterministe ($d(P_i,P_j) > C(c_i,c_j)$). Dans de ces cas, $E_{ij}$ dénote la différence entre $d(P_i,P_j)$ et $C(c_i,c_j)$, c'est-à-dire $E_{ij} = d(P_i,P_j) - C(c_i,c_j)$.

- Le minimum, le maximum et la moyenne arithmétique des $E_{ij}$.

L'analyse des résultats de cette évaluation montre que le nombre NB_CBR_ND dépend du quantificateur linguistique utilisé dans l'évaluation de la similarité entre les projets. Autrement dit, NB_CBR_ND dépend de $\alpha$. Dans les deux cas des mesures C_A et C_R, NB_CBR_ND est monotone croissant en fonction de $\alpha$. Ceci est dû au fait que nos mesures de similarité sont monotones décroissantes en fonction de $\alpha$ ( $d_a(P_i,P_j) \geq d_{a'}(P_i,P_j)$ $a \leq a'$). Si nous considérons le cas du quantificateur linguistique *all* (la ligne *min* du tableau 4.5) pour évaluer la similarité globale entre les projets et la mesure C_A pour évaluer celle entre leurs coûts, nous constatons que dans 77 cas (seulement 1,94%) l'affirmation est non-déterministe. Par conséquent, nous avons opté, dans un premier temps, pour l'utilisation de la version déterministe de l'affirmation *Similar project have similar costs* dans *Fuzzy Analogy.*

La formulation d'une estimation au coût d'un projet P dans la version déterministe consiste tout d'abord à chercher tous les projets historiques $P_i$ qui sont étroitement similaires à P, ensuite, déterminer pour chaque $P_i$, l'ensemble des valeurs similaires au coût de $P_i$ avec un degré de similarité supérieur ou égal à $d(P,P_i)$, $E_i(P)$:

$$E_i(P) = \{c / C(c,c_i) \geq d(P,P_i)\} \qquad \text{(Équation 4.11)}$$

où $c_i$ est le coût de $P_i$. Les valeurs possibles au coût du P sont donc données par l'ensemble E(P):

$$E(P) = \bigcap_i E_i(P) \qquad \text{(Équation 4.12)}$$

En appliquant cette procédure de la version déterministe au cas de la base COCOMO'81, nous avons constaté que, dans la plupart des cas, l'ensemble E(P) était vide. En effet, un projet P peut être similaire à plusieurs projets $P_i$ mais les coûts des $P_i$ sont totalement différents. Par exemple, le projet $P_9$ de la base du COCOMO'81 est évaluée être similaire à $P_{27}$ et $P_{56}$ avec respectivement les degrés 0,30 et 0,19. Cependant, il n'existe aucune valeur numérique qui est à la fois similaire, selon C_A ou C_R, au coût de $P_{27}$ (88 K$) aussi bien qu'au coût de $P_{56}$ (985 K$). Pour remédier à cette situation, Dubois et al. suggèrent que les

mesures de similarités, utilisées dans la version déterministe, doivent satisfaire deux propriétés relatives à la cohérence des deux parties (prémisse et conséquence) des règles floues suivantes (Dubois et al., 1999):

$$P_j \in E_i(P) \Rightarrow c_j \in C_i(c)$$

Ces deux propriétés garantissent que l'ensemble E(P), associé à un projet P, ne peut être vide. Nous ne sommes pas convaincu par leur suggestion puisqu'elle peut mener à des mesures de similarité (entre projets ou entre leurs coûts) qui contredisent notre intuition concernant l'attribut *similarité*. Ainsi, nous avons opté pour la version non-déterministe afin de gérer les incertitudes au niveau des estimations fournies par *Fuzzy Analogy*.

### 4.5.3 Formulation de la version non-déterministe du raisonnement par analogie en estimation des coûts

La version non-déterministe de l'affirmation *Similar software projects have similar costs* peut être reformulée comme suit: *Similar oftware projects have* ***possibly*** *similar costs*, c'est-à-dire, si un projet P est similaire à $P_i$, il est possible que le coût de P soit similaire à celui de $P_i$. Dubois et al. proposent de représenter cette affirmation de la version non-déterministe par la règle de possibilité suivante (Dubois et al., 1999):

*Autant P est similaire à $P_i$, autant il est possible que le coût de P soit similaire à celui de $P_i$*

Cette règle exprime qu'il est possible, avec un degré au moins égal à $d(P,P_i)$, que le coût de P soit similaire à celui de $P_i$. En plus, toute autre valeur numérique similaire au coût de $P_i$ est aussi possible, avec au moins le même degré, qu'elle soit similaire au coût du projet P. La modélisation de la règle de possibilité ci-dessus, en utilisant la conjonction pour l'implication, donne l'ensemble flou de toutes les valeurs possibles au coût du P:

$$p_{cost(P_i)}(c) = min(d(P,P_i), C\_R_{cost(P_i)}(c)) \qquad \text{(Équation 4.13)}$$

Dans le cas où la base de données contient N projets logiciels, chaque projet $P_i$ génère un ensemble de valeurs possibles au coût du P selon l'équation 4.13. Ces ensembles flous sont

combinés par l'opérateur *max* (opération de disjonction) afin d'obtenir l'ensemble flou $C_p$ contenant toutes les valeurs possibles du coût du projet P:

$$C_P(c) = \max_i(p_{cost(P_i)}(c)) \qquad \text{(Équation 4 .14)}$$

La fonction d'appartenance à l'ensemble $C_p$ représente la distribution de possibilités du coût de P. Il sera utilisée par *Fuzzy Analogy*, pour évaluer le degré d'incertitude associé à une estimation du coût du projet P. Par exemple, la figure 4.7 montre la distribution de possibilités associée au coût du projet $P_{45}$ de la base COCOMO'81. $P_{45}$ est similaire aux projets $P_{42}$, $P_{43}$, $P_{44}$, et $P_{46}$ avec des degrés respectivement égaux à 0,38, 0,40, 0,38 et 0,36. La distribution de possibilités est définie seulement pour les valeurs réelles similaires à au moins une des quatre valeurs représentant les coûts de $P_{42}$, $P_{43}$, $P_{44}$, et $P_{46}$; pour les autres valeurs, la distribution de possibilité est inconnue.

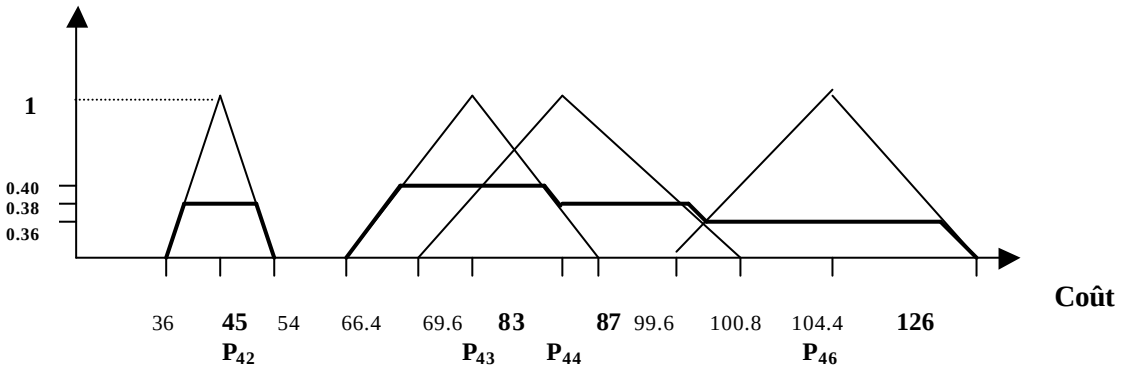



**Figure 4.7** Exemple d'une distribution de possibilités associée au coût du projet $P_{45}$ du COCOMO'81.

La détermination d'une estimation numérique au coût d'un projet P à partir de sa distribution de possibilités utilise, en général, des techniques de *defuzzufication* telles que la technique du centre de gravité et la technique *mean-of-maxima*. Ces techniques ne sont pas forcément appliquées à tout l'ensemble flou dont la fonction d'appartenance est la distribution de possibilités du coût. Elles peuvent être appliquées seulement à une partie de cet ensemble flou. Par exemple, dans le cas du projet $P_{45}$, il est fort probable que son coût réel soit dans l'intervalle [66,4, 152,2]. Ainsi, l'application de la technique du centre de gravité à cet

intervalle génère une estimation égale à 100 K$ au coût du $P_{45}$; l'utilisation de notre technique dite *crisp* de l'équation 4.9 génère une estimation égale à 68 K$. Le coût réel de $P_{45}$ est 106 K$.

L'importance de la gestion des incertitudes dans *Fuzzy Analogy* découle du fait qu'elle permet l'évaluation des risques associés à ces incertitudes. En effet, la fonction de distribution de possibilités du coût d'un projet P peut être utilisée pour évaluer les risques d'une estimation choisie parmi toutes les valeurs possibles du coût de P. Par exemple, pour le cas du projet $P_{45}$, si nous adoptons une estimation optimiste à son coût, à savoir 45 K$, et s'il s'avère après que $P_{45}$ a nécessité 126 K$ dans son développement, nous aurons donc un déficit de 81 K$ afin d'achever le développement de $P_{45}$. Kitchenham et Linkman proposent l'utilisation d'une stratégie d'assurance des projets logiciels telle que celle utilisée par les compagnies d'assurance de voitures. Par conséquent, nous pouvons affecter à chaque projet logiciel, un contingent qui dépend du risque et de la possibilité de ce risque:

$$Contingent = (E_2 - E_1) \times P_2 \qquad \text{(Équation 4.15)}$$

où $E_1$ est l'estimation adoptée, $E_2$ est l'estimation possible et $P_2$ est la possibilité d'avoir l'estimation $E_2$. Dans le cas du projet $P_{45}$, si l'estimation adoptée à son coût est égale à 45 K$, le contingent est égal à 29,06 K$ pour un risque d'avoir un coût réel de 126 K$. Dans certains cas, la fonction de distribution de possibilités du coût d'un projet peut être non-informative. C'est le cas d'un projet logiciel similaire à plusieurs projets ayant des coûts totalement différents (Fig. 4.8).

Afin de réduire la probabilité d'avoir le cas d'une distribution de possibilités du coût non-informative, nous proposons l'utilisation des valeurs linguistiques à la place des valeurs numériques, pour l'évaluation du coût d'un projet logiciel. Ainsi, l'ensemble flou de toutes les valeurs possibles au coût d'un projet P en tenant en compte d'un seul projet historique $P_i$ est:

$$p_{P_i}(c) = \max_k \min(d(P, P_i), m_{A_k}(c)) \qquad \text{(Équation 4.16)}$$

où $A_k$ sont les ensembles flous associés aux valeurs linguistiques du coût.

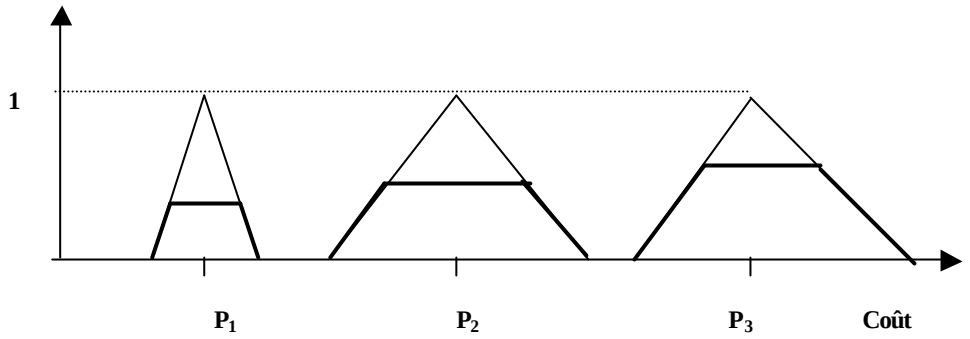



**Figure 4.8** Exemple d'une distribution de possibilités non-informative du coût d'un projet P similaire à trois projets $P_1$, $P_2$ et $P_3$.

La distribution de possibilités du coût de P en considérant tous les projets historiques $P_i$, $C_p$, s'obtient en combinant toutes les fonctions de l'équation 4.16 par l'opérateur *max*. Par exemple, supposons que le coût d'un projet est évalué par trois valeurs linguistiques: *faible*, *moyenne* et *élevé*. Nous pouvons constater que, dans le cas d'un projet P similaire à deux projets $P_1$ et $P_2$ avec des degrés respectivement égaux à 0,5 et 0,7, il n'y a aucune région où la distribution de possibilités du coût de P est inconnue; ceci bien que les coûts de $P_1$ et $P_2$ soient totalement différents.

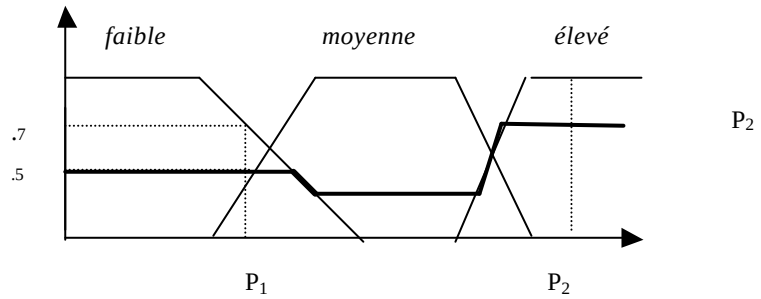



**Figure 4.9** Exemple d'utilisation des valeurs linguistiques pour éviter les intervalles où une distribution de possibilités peut être inconnue.

## 4.6 DISCUSSION ET CONCLUSION

Dans ce chapitre, nous avons présenté une nouvelle version floue de l'estimation des coûts par analogie: *Fuzzy Analogy*. Cette approche permet la tolérance des imprécisions tout au long du processus d'estimation. Ainsi, dans sa première étape d'Identification des projets logiciels, elle représente les valeurs linguistiques, décrivant les projets logiciels par des ensembles flous plutôt que par des ensembles classiques. Dans sa deuxième étape de l'Évaluation de la similarité entre les projets logiciels, elle adopte de nouvelles mesures de similarité basées sur les techniques d'agrégations floues et les quantificateurs linguistiques. Ces mesures de similarité ont été validées selon une approche axiomatique utilisant quatre axiomes. Dans sa troisième étape d'Adaptation, elle utilise la valeur linguistique *étroitement similaire* dans le choix des projets logiciels qui seront utilisés pour déduire une estimation du coût du nouveau projet. *Fuzzy Analogy* a été validée en utilisant les projets logiciels du COCOMO'81. La précision de ses estimations a été comparée avec celles de trois autres techniques: COCOMO'81 Intermédiaire, *fuzzy* COCOMO'81 Intermédiaire et l'Analogie classique. Les résultats de cette étude comparative avantagent notre approche *Fuzzy Anlogy* sur les trois autres techniques.

*Fuzzy Analogy* permet aussi la gestion des incertitudes au niveau de l'estimation des coûts en utilisant la théorie de possibilité de Zadeh. Ainsi, elle fournit un ensemble de valeurs estimées du coût d'un projet avec une fonction de distribution des possibilités indiquant pour chaque valeur son degré de possibilité pour qu'elle soit la valeur réelle du coût. Cette fonction de distribution peut être aussi utilisée pour la gestion des risques attachés aux incertitudes des estimations.

Par cela, *Fuzzy Analogy* satisfait les deux premiers critères d'intelligence: la tolérance des imprécisions et la gestion des incertitudes au niveau des estimations fournies. Il lui reste donc l'intégration de l'apprentissage pour satisfaire le dernier critère d'intelligence. Le chapitre suivant présente et discute la stratégie que nous adoptons pour cet objectif.

# CHAPITRE V

# INTÉGRATION DE L'APPRENTISSAGE DANS L'ESTIMATION DES COÛTS PAR *FUZZY ANALOGY*

Dans le chapitre précédent, nous avons présenté et discuté du processus d'intégration des deux premières caractéristiques de l'intelligence, à savoir la tolérance des imprécisions et la gestion des incertitudes au niveau de l'estimation des coûts dans notre approche *Fuzzy Analogy*. Le présent chapitre porte sur la stratégie que nous adoptons pour intégrer l'apprentissage, la troisième caractéristique de l'intelligence, dans notre approche *Fuzzy Analogy*. Cette stratégie consiste essentiellement en la mise à jour des différents paramètres de *Fuzzy Analogy*, tels que les définitions des valeurs linguistiques, la définition du quantificateur linguistique et la combinaison des attributs utilisés pour la description des projets logiciels. Dans une deuxième partie de ce chapitre, nous étudions l'utilisation des réseaux de neurones pour incorporer un apprentissage automatique dans *Fuzzy Analogy*. Nous traitons particulièrement du cas du réseau RBFN (*Radial Basis Function Network*s). L'objectif de l'utilisation d'un réseau RBFN est de faciliter le processus de mise à jour des différents paramètres de *Fuzzy Analogy*.

## 5.1 INTRODUCTION

L'apprentissage est une qualité des humains, et il est certainement derrière une large part de notre intelligence. Il nous permet de nous adapter, d'apprendre et d'acquérir de nouvelles connaissances à partir de notre environnement. Ainsi, l'apprentissage est considéré comme une condition nécessaire que les systèmes dits intelligents doivent satisfaire (Negnevitsky, 2002). En effet, les problèmes pour lesquels l'intelligence artificielle doit faire face pour apporter des solutions satisfaisantes, sont ceux dont les détails et les contraintes ne peuvent être identifiées clairement. Par conséquent, les solutions proposées, pour un échantillon de

cas du domaine étudié, doivent être capables de répondre aux besoins et aux spécificités des cas futurs et imprévus. Par exemple, un système qui gère les déplacements d'un robot dans une maison doit être capable d'apprendre afin d'ajuster les déplacements du robot en fonction des changements et des aménagements qu'un individu pourra faire dans sa maison.

La nécessité de l'apprentissage dans un modèle d'estimation des coûts provient du fait que l'industrie des logiciels est incessamment en évolution. En effet:

- On ne développe plus le même type d'application. Aujourd'hui, l'emphase est mise sur les applications distribuées et les applications Web alors que, dans les décades précédentes, elle était mise sur les applications mono-poste classiques.

- La complexité des applications informatiques actuelles est de plus en plus élevée du fait de la diversité des besoins accrus en matière d'informatisation de la plupart des services dans les sociétés modernes.

- Le personnel impliqué dans le développement et/ou la maintenance de logiciels est de plus en plus qualifié.

- On utilise de nouvelles méthodes de développement et/ou de maintenance.

Par conséquent, l'apprentissage permettra à un modèle d'estimation de se reconfigurer et de s'adapter aux nouveaux changements survenus dans son environnement, afin d'ajuster ses estimations aux coûts des nouveaux projets. Il y a deux autres avantages à la caractéristique d'apprentissage en estimation des coûts: 1) L'apprentissage est un moyen efficace pour améliorer la précision d'un modèle d'estimation des coûts; 2) l'apprentissage peut être utilisé pour calibrer et pour adapter un modèle d'estimation des coûts, développé dans un environnement, vers un environnement différent.

## 5.2 MISE À JOUR DES PARAMÈTRES DE *FUZZY ANALOGY*

La technique *Fuzzy Analogy* utilise plusieurs paramètres qui sont définis en fonction des besoins de l'environnement concerné. Ainsi, les définitions de ces paramètres dépendent de l'état de cet environnement. Par conséquent, notre modèle d'estimation doit permettre à ses

utilisateurs la mise à jour de ses paramètres afin qu'il puisse être continuellement en cohérence avec son environnement. De ce fait, nous intégrons dans le logiciel prototype F_ANGEL plusieurs fonctionnalités qui permettent la mise à jour de ces paramètres. En effet, F_ANGEL permet de :

- Supprimer les attributs qui ne sont plus significatifs pour l'estimation des coûts de projets logiciels.
- Ajouter de nouveaux attributs significatifs pour l'estimation des coûts.
- Modifier les informations sur un attribut particulier (voir Annexe B pour les écrans de F_ANGEL relatifs à ces fonctionnalités):

    a) *Ajout de nouvelles valeurs linguistiques*. par exemple, un attribut logiciel tel que l'expérience des programmeurs pourra être évalué durant une première période par seulement trois valeurs linguistiques: *bas*, *moyen* et *élevé*. Avec l'ancienneté cumulée par les programmeurs, d'autres valeurs linguistiques telles que *très élevé* et *extra-élevé* seront nécessaires pour une meilleure évaluation de l'expérience des programmeurs.

    b) *Suppression d'anciennes valeurs linguistiques*. par exemple, du fait qu'on demande de plus en plus des logiciels très fiables, la valeur linguistique *très bas* ne serait plus utile pour l'évaluation de la fiabilité d'un logiciel. Il est donc préférable de la supprimer de l'échelle d'évaluation de la fiabilité logicielle.

    c) *Modification de la fonction d'appartenance associée à une valeur linguistique*. une fonction d'appartenance à un ensemble flou représente notre compréhension de la valeur linguistique associée à cet ensemble flou. Ainsi, cette fonction est sujette à des modifications pour répondre à de nouveaux besoins qui surviennent dans l'environnement concerné. Par exemple, dans un organisme la fiabilité d'un logiciel serait évaluée par *élevé* si le nombre de défaillances logicielles est de 6 à 10 par mois. Pour

répondre à des exigences externes (normes de qualité par exemple), l'organisme pourra redéfinir la valeur linguistique *élevé* par 3 à 7 défaillances par mois.

d) *Changement du poids associé à l'attribut*: par exemple, un organisme qui développe des applications de gestion mono-poste affectera un poids faible à l'attribut *sécurité du logiciel*. Si l'organisme investit dans le développement des applications de gestion distribuées, il doit affecter un poids supérieur à celui du cas des applications mono-poste, à l'attribut *sécurité du logiciel*.

- Modifier la définition de la valeur linguistique *étroitement similaire* utilisée dans l'étape d'Adaptation afin de prendre en considération les nouvelles exigences sur la sélection des projets qui vont contribuer à l'estimation du coût d'un nouveau projet.

- Mise à jour de la base de projets logiciels historiques en permettant à l'utilisateur de modifier les valeurs des attributs d'un projet, supprimer un projet de la base quant il ne représente plus des ressemblances avec les nouveaux projets et ajouter un nouveau projet à la base quand il est achevé afin de maintenir la cohérence de la base avec son environnement (voir Annexe B pour l'écran de F_ANGEL relatif à cette fonctionnalité).

- Modifier la définition du quantificateur linguistique utilisé dans l'évaluation de la similarité globale entre les projets logiciels. En effet, ce quantificateur représente la combinaison adoptée dans l'environnement pour l'évaluation de la similarité globale entre deux projets à partir de leurs similarités individuelles. Cette combinaison indique le nombre d'attributs utilisés ainsi que leurs contributions dans l'évaluation de la similarité globale entre deux projets logiciels. L'utilisation de notre approche *Fuzzy Analogy* dans un environnement requiert donc le choix d'un quantificateur linguistique approprié à l'environnement concerné. Le choix de ce quantificateur n'est pas souvent facile. En effet, le quantificateur linguistique choisi doit, d'une part, représenter convenablement les caractéristiques de l'environnement et, d'autre part, améliorer la performance de notre modèle. Nous présentons, dans ce qui suit, notre

réflexion concernant le choix du quantificateur linguistique approprié à un environnement.

Dans la littérature, Yager a identifié trois types de quantificateurs linguistiques (Yager, 1996):

- Quantificateurs monotones croissants (Fig. 5.1a): Ce sont les quantificateurs linguistiques dont la valeur obtenue lors de la combinaison des critères, croît en fonction du nombre de critères satisfaits. Des exemples de ces quantificateurs sont *all*, *most*, *many*, et *at-least* $a$. Ce genre de quantificateurs peuvent être représentés par un ensemble flou ayant comme univers de discours l'intervalle [0,1]. La fonction d'appartenance, *Q*, à cet ensemble flou doit satisfaire les trois conditions suivantes:

  a) Q(0)=0,
  b) Q(1)=1, et
  c) $Q(x) \geq Q(y)$ si $x \geq y$.

- Quantificateurs monotones décroissants (Fig. 5.1b): Ce sont les quantificateurs linguistiques dont la valeur obtenue lors de la combinaison des critères, décroît en fonction du nombre de critères satisfaits. Des exemples de ces quantificateurs sont *few* et *at-most* $a$. Ce genre de quantificateurs peuvent être représentés par un ensemble flou ayant comme univers de discours l'intervalle [0,1]. La fonction d'appartenance, *Q*, à cet ensemble flou doit satisfaire les trois conditions suivantes:

  a) Q(0)=1,
  b) Q(1)=0, et
  c) $Q(x) \geq Q(y)$ si $x \leq y$.

- Quantificateurs non-monotones centrés (Fig. 5.1c): Ce sont les quantificateurs linguistiques dont la valeur obtenue lors de la combinaison des critères, croît quand le nombre de critères satisfaits est au voisinage d'une valeur $a$ donnée. Des exemples de ces quantificateurs sont ceux de la forme *about-*$a$. Ce genre de quantificateurs peuvent être représentés par un ensemble flou ayant comme univers de discours

l'intervalle [0,1]. La fonction d'appartenance, *Q*, à cet ensemble flou doit satisfaire les trois conditions suivantes:

a) Q(0)=0,
b) Q(1)=0, et
c) Il existe deux valeurs a et b de l'intervalle [0,1] telles que:
    1) $Q(x) \leq Q(y)$ si $x \leq y$ avec $y < a$
    2) $Q(x)=1$ si $x \in [a,b]$
    3) $Q(x) \geq Q(y)$ si $x \leq y$ avec $y > b$

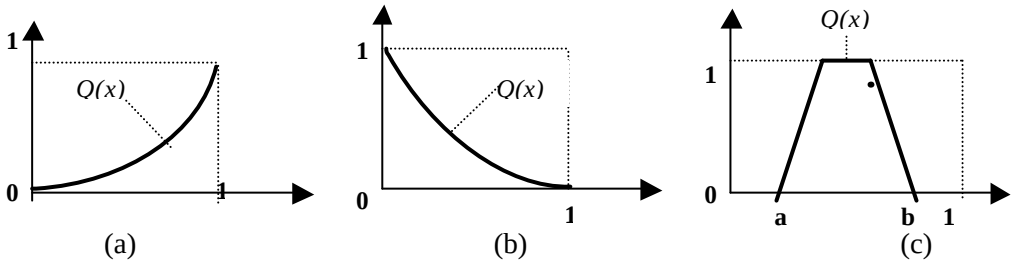



**Figure 5.1** Exemples de fonctions d'appartenance des trois types de quantificateurs linguistiques. (a) Quantificateurs linguistiques monotones croissants; (b) Quantificateurs linguistiques monotones décroissants; (c) Quantificateurs linguistiques non-monotones centrés.

Dans notre approche *Fuzzy Analogy*, l'évaluation de la similarité globale entre deux projets utilise un quantificateur linguistique monotone croissant. En effet, intuitivement, la similarité globale croît en fonction du nombre de similarités individuelles satisfaites. Par exemple, si le quantificateur linguistique utilisé est *at-least 4*, la similarité globale entre deux projets ayant six similarités individuelles satisfaites doit être supérieure ou égale à celle entre deux projets ayant seulement cinq, parmi les six, similarités individuelles qui sont satisfaites. Dans nos expérimentations sur la base COCOMO'81, nous avons utilisé des quantificateurs linguistiques croissants (RIM) dont les fonctions d'appartenance sont de la forme $Q(x)=x^a$ et cela pour deux raisons principales: premièrement, la base de projets COCOMO'81 ne contient pas les informations nécessaires pour identifier le quantificateur linguistique croissant approprié à l'environnement du COCOMO'81; deuxièmement, la fonction $Q(x)=x^a$ peut se rapprocher avec une grande précision de plusieurs types de fonctions d'appartenance

associées aux quantificateurs linguistiques monotones croissants, en choisissant une valeur adéquate de $a$.

Jusqu'à présent, l'outil F_ANGEL permet la modification de la fonction $Q(x)=x^a$ en allouant à l'estimateur le choix d'une valeur convenable de $a$. Ce choix de $a$ est subjectif. Ainsi, l'estimateur pourra expérimenter plusieurs valeurs de $a$ afin de retrouver celle qui répond aux besoins de son environnement. Cette stratégie nécessite une expertise et pourra entraîner des délais d'attente considérables avant de retrouver la valeur convenable de $a$. Afin de guider l'estimateur dans son choix de $a$, nous suggérons l'intégration d'une technique heuristique utilisant le concept de la programmation génétique. L'objectif de cette technique est de permettre à F_ANGEL de proposer à l'estimateur un ensemble de valeurs possibles de $a$ en utilisant son historique des valeurs $a$ qui ont souvent mené à des estimations précises. Cet ensemble de valeurs possibles de $a$ est déterminé par un apprentissage supervisé. Ainsi, pour chaque projet historique $P_i$, nous déterminons la valeur $\overline{a_i}$ qui permet à F_ANGEL de générer l'estimation la plus proche de la valeur réelle du coût de $P_i$. La valeur $\overline{a_i}$ est déterminée par le programme génétique suivant:

1- Générer aléatoirement une population initiale de valeurs $a_i$;

2- Répéter les sous-étapes suivantes jusqu'à avoir une MRE acceptable ou un nombre d'itérations est terminé:

   a. Pour chaque $a_i$, F_ANGEL évalue la précision de son estimation au coût du $P_i$ en utilisant l'indicateur MRE.
   b. Sélectionner parmi tous les $a_i$ ceux ayant les meilleures MRE.
   c. Reproduire une nouvelle population de $a_i$ en appliquant les deux opérations de croisement et de mutation.
   d. Remplacer l'ancienne population par la nouvelle population du sous- étape c.

3- Choisir comme valeur de $\overline{a_i}$ la valeur $a_i$ ayant la meilleure MRE de la population finale.

Après la détermination de toutes les valeurs $\overline{a}_i$, F_ANGEL sélectionne seulement celles ayant les meilleurs scores de MRE et les proposent, selon un ordre décroissant, à l'estimateur afin de les utiliser dans le cas d'un nouveau projet P. Une fois ce nouveau projet P achevé, c'est-à-dire lorsque la valeur réelle de son coût est connue, F_ANGEL détermine la valeur $\overline{a}_i$ associée en utilisant le même programme génétique ci-dessus (Fig. 5.2). Dans le cas de la base de projets du COCOMO'81, la validation que nous avons menée pour évaluer la précision des estimations de *Fuzzy Analogy* (Tab. 4.3) avantage l'utilisation d'une valeur de $a$ supérieure ou égale à 30. Par conséquent, il n'était pas nécessaire, dans le cas du COCOMO'81, d'appliquer la procédure de la figure 5.2 pour retrouver le meilleur $a$.

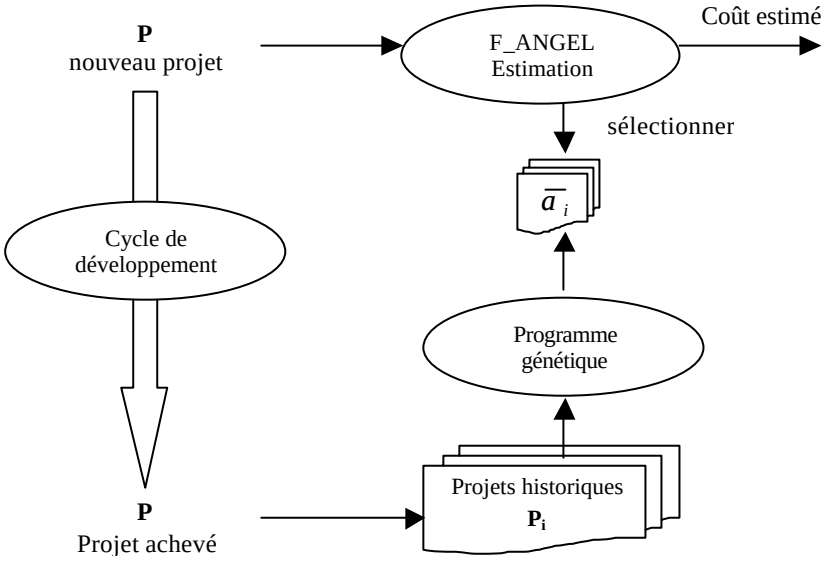



**Figure 5.2** Procédure d'apprentissage de F_ANGEL pour la détermination du quantificateur linguistique à utiliser dans l'estimation du coût d'un nouveau projet P.

## 5.3 APPRENTISSAGE PAR LES RÉSEAUX DE NEURONES

Les réseaux de neurones artificiels (ANN) sont une réalisation de l'approche connexionniste des sciences cognitives. Ils regroupent un certain nombre de modèles dont l'intention est d'imiter certaines des fonctions du cerveau humain en s'inspirant de certaines de ses structures de base. Historiquement, les origines des ANN datent de 1943 suite à la publication des travaux de McCulloch et Pitts où ils étudièrent un ensemble de neurones

formels interconnectés et montrèrent leurs capacités à calculer certaines fonctions logiques. En 1949, Hebb souligna l'importance du couplage synaptique dans le processus d'apprentissage. C'est en 1958 que Rosenblatt décrivit le premier modèle opérationnel de réseaux de neurones, mettant en œuvre les idées de McCulloch, Pitts et Hebb: le Perceptron. Ce modèle suscita beaucoup de recherches, et sans doute beaucoup d'espoir. Malheureusement, le manque de machines puissantes capables de faire des expérimentations avec les réseaux de neurones artificiels ainsi que la critique pertinente de Minsky et Papert démontrant l'insuffisance du Perceptron avec une seule couche pour la résolution des problèmes non-linéaires avaient incité la communauté en intelligence artificielle à s'intéresser plus aux systèmes experts et à abandonner ainsi l'alternative neuronale.

Au début des années 1980, les réseaux de neurones artificiels vont connaître un nouvel essor (*the rebirth of neural networks*) grâce à l'évolution des technologies de l'ordinateur, aux progrès de la neuroscience et aux limites de l'approche des systèmes experts surtout au niveau de leur fragilité et de leur difficulté à s'adapter aux changements dans les environnements. Ainsi, le domaine des ANN connaîtra quelques découvertes intéressantes (Negnevitsky, 2002): Grossberg (1980) avait proposé les réseaux de neurones *Self-organisation*; Hopfield (1982) avait introduit des réseaux de neurones avec feedback (mémoires auto-associatives); Rumelhart et McClelland (1986) ont développé le célèbre algorithme de la Rétro-propagation (*Backpropagation*) pour l'apprentissage dans un Perceptron multi-couches. Les réseaux de neurones ont largement profité de ces découvertes pour s'imposer au sein du domaine de l'intelligence artificielle comme étant l'outil préféré pour implanter l'apprentissage dans les systèmes intelligents.

Les réseaux de neurones se distinguent de la plupart des approches de l'intelligence artificielle par le fait qu'ils intègrent des mécanismes d'apprentissage automatique dans les solutions qu'ils proposent. Le célèbre algorithme de la Rétro-propagation est un exemple typique qui a été utilisé avec succès dans divers domaines (Hertz, Krogh et Palmer, 1991; Hayken, 1994). L'apprentissage dans les réseaux de neurones artificiels trouve ses origines dans les principes de l'approche connexionniste en sciences cognitives bien que le connexionnisme avait lui-aussi profité de l'existence d'une théorie des réseaux de neurones. En effet, le connexionnisme s'intéresse plus, par contraste au symbolisme, à la construction

du processus qui va faire la tâche et non pas à la tâche en elle-même. Les autres techniques de l'intelligence artificielle ne donnent pas beaucoup d'importance à l'apprentissage. Par conséquent, son implantation dans les solutions qu'elles proposent, s'avère insuffisante ou parfois même impossible:

- Les systèmes experts se basent généralement sur une base de règles. Ces règles reflètent l'état actuel de notre compréhension du problème étudié. Par conséquent, un mécanisme d'apprentissage, dans un système expert, doit nécessairement mettre à jour la base de règles (ajouter, supprimer ou modifier certaines règles) afin d'adapter le système à son environnement. Vu la complexité d'une base de règles, contenant souvent un nombre très élevé de règles, il est presque impossible, même manuellement, de mettre à jour une telle base. En effet, il faut étudier les impacts de chaque opération de mise à jour sur la cohérence et la consistance de la base de règles (redondance, contradiction, signification, etc.). La conception d'une base de règles, relative à un domaine complexe, est déjà un grand défi pour les systèmes experts. Donc, il ne faut pas s'attendre à ce que sa maintenance (mécanisme d'apprentissage) soit une tâche facile.

- Les méthodes de régression linéaire statistiques n'intègrent pas de mécanismes d'apprentissage. Tout ce qu'on peut faire est de reconstruire la fonction linéaire une fois qu'un nouveau patron est disponible. Ce genre d'apprentissage est insuffisant pour résoudre des problèmes complexes.

- Les arbres de décision sont générés à partir d'un ensemble de patrons. Ils n'offrent pas de mécanismes d'apprentissage automatiques. Une fois qu'un nouveau patron est disponible, il faut régénérer l'arbre afin de prendre en considération les nouvelles connaissances contenues dans le nouveau patron.

Il y a deux types fondamentaux d'apprentissage par les réseaux de neurones (Hertz, Krogh et Palmer, 1991):

- L'apprentissage supervisé où le réseau compare sa sortie finale avec la sortie attendue et rétro-propage l'erreur calculée afin d'ajuster ses poids (*learning with a*

*teacher*). Une autre forme de l'apprentissage supervisé est l'apprentissage renforcé (*reinforcement learning*) où le réseau reçoit un feedback de son environnement par rapport à l'évaluation de sa sortie mais sans recevoir d'instructions pour ajuster ses poids (learning with a critic).

- L'apprentissage non-supervisé où le réseau ne reçoit aucun feedback de l'environnement au sujet de sa sortie. Le réseau doit ajuster ses poids dépendamment des caractéristiques, des régularités, des corrélations, ou des catégories qu'il découvre dans les patrons d'apprentissage. L'apprentissage non-supervisé ne peut être efficace que dans les cas où les patrons d'apprentissage présentent des redondances (*redundancy provides knowledge*)

Chaque type d'apprentissage (supervisé ou non-supervisé) a des avantages et des inconvénients. L'apprentissage supervisé est facile à utiliser; il est plausible pour l'approximation de fonctions. Cependant, il est un grand consommateur de temps de calcul. L'apprentissage non-supervisé peut être utile dans le cas où les valeurs réelles du problème étudié ne sont pas disponibles; il est adapté aux tâches de mémoires associatives et de catégorisation. Cependant, il n'est pas facile d'implanter et de contrôler sa convergence. Certains types de réseaux de neurones combinent les deux catégories d'apprentissage. C'est le cas des réseaux RBFN de Moody et Darken où dans la couche cachée on utilise un apprentissage non-supervisé (apprentissage compétitif), et dans la couche de sortie on utilise un apprentissage supervisé (descente de gradient) (Moody et Darken, 1989).

De ce fait, nous étudions l'équivalence entre notre approche *Fuzzy Analogy* et les réseaux de neurones afin d'incorporer un apprentissage automatique dans F_ANGEL. L'idée principale est de modifier automatiquement les différents paramètres de *Fuzzy Analogy* (définitions des valeurs linguistiques, définition de la valeur linguistique *étroitement similaire* et définition du quantificateur linguistique) en utilisant le réseau de neurones équivalent à *Fuzzy Analogy*. En plus, il y a deux autres avantages à l'établissement d'une équivalence entre *Fuzzy Analogy* et un type particulier de réseaux de neurones: premièrement, nous pouvons appliquer tout ce qui est découvert par l'un des modèles à l'autre; deuxièmement, nous pouvons fournir une interprétation compréhensible à la connaissance contenue dans le réseau de neurones à travers

sa représentation par des mécanismes d'analogie. Ainsi, nous évitons la critique principale, portant sur le caractère opaque de l'approche neuronique, qui a souvent empêché cette approche d'être acceptée comme une pratique usuelle en estimation des coûts. En effet, les réseaux de neurones ne fournissent pas une interprétation ou une explication à leur processus de raisonnement. Par conséquent, il est très difficile d'évaluer leur fiabilité. L'inconvénient d'une modélisation *boîte noire* de la relation exprimant l'effort en fonction d'un ensemble de conducteurs de coût se manifeste quand la sortie générée est différente de la sortie attendue; dans ce cas, les causes de ce genre de comportement ne peuvent être expliquées.

Dans un premier temps et avant d'étudier une équivalence directe entre notre approche *Fuzzy Analogy* et un réseau de neurones particulier, nous proposons d'étudier l'équivalence entre les réseaux de neurones et les systèmes à base de règles floues (Fuzzy Rule-Based Systems -FRBS-). En effet, il existe dans la littérature une variété de méthodes qui permettent la transformation d'un réseau de neurones en un système de règles floues (Jang et Sun, 1992; Buckley, Hayashi et Czogala, 1993; Benitez, Castro et Requena, 1997). Les objectifs principaux de cette étude préalable sont, d'une part, de fournir une interprétation compréhensible de la connaissance contenue dans l'architecture et les synapses d'un réseau de neurones à travers l'interprétation des règles floues Si-Alors obtenues et, d'autre part, l'éventualité d'incorporer ces règles floues dans les deux dernières étapes de *Fuzzy Analogy* (Evaluation de la similarité et l'Adaptation). Ainsi, l'équivalence entre les réseaux de neurones et notre approche *Fuzzy Analogy* serait établie indirectement par l'utilisation d'un système FRBS. Cette étude préalable considère le cas d'un Perceptron à trois couches avec l'algorithme de Rétro-propagation (Backpropagation).

## 5.4 INTERPRÉTATION D'UN PERCEPTRON À TROIS COUCHES EN ESTIMATION DES COÛTS

Le Perceptron multi-couches avec l'algorithme de Rétro-propagation pour l'apprentissage est le plus utilisé en estimation des coûts parmi tous les autres types de réseaux de neurones (Samson, Ellison et Dugard, 1993; Serluca, 1995; Srinivasan et Fisher, 1995; Jorgersen, 1995; Hughes, 1996; Wittig et Finnie, 1997; Shukla, 2000). Cependant, la plupart de ces travaux de recherche se sont intéressés à l'évaluation de la précision des estimations de l'approche neuronique en comparaison avec celles des autres techniques telles que la

régression linéaire, les arbres de décision et le *Case-Based Reasoning* (Tab. 1.1). Dans cette section, nous étudions l'interprétation d'un Perceptron à trois couches en utilisant les projets du COCOMO'81.

Le choix d'une architecture de trois couches est justifié par le fait que la méthode que nous utiliserons pour générer les règles floues Si-Alors à partir du Perceptron, exige que le réseau soit formé de trois couches et que les fonctions d'activation soient la fonction Sigmoïde pour la couche cachée et la fonction Identité pour la couche de sortie (Équation 1.14). Notre réseau de neurones a 13 entrées (facteurs du COCOMO'81) et une seule sortie (effort). Toutes les entrées ainsi que la sortie du réseau sont numériques. Toutes les entrées sont normalisées pour améliorer la performance du processus d'apprentissage du réseau (Hertz, Krogh et Palmer, 1991). L'apprentissage du réseau est réalisé en présentant les données plusieurs fois au réseau; elle utilise l'algorithme de rétro-propagation avec un taux d'apprentissage de 0,03 et une erreur maximale de l'ordre de $10^{-5}$. Les poids des différentes connexions sont initialisés avec des valeurs aléatoires très petites. D'après l'architecture de notre réseau, l'effort (sortie) est calculé par:

$$Effort = \sum_{j=1}^{h} z_j b_j \quad avec \quad z_j = f(\sum_{i=1}^{n} w_{ij} x_i) \qquad \text{(Équation 5.1)}$$

où $f$ est la fonction Sigmoïde, $w_{ij}$ sont les poids des connexions entre la couche des entrées et la couche cachée et les $b_j$ sont les poids des connexions entre la couche cachée et la couche de sortie.

### 5.4.1 Discussion des résultats de l'évaluation de la précision des estimations du Perceptron à trois couches

Dans cette section, nous présentons et nous discutons des résultats obtenus de l'application de notre réseau de neurones aux projets logiciels de la base de données COCOMO'81. Nous avons développé un logiciel prototype en langage C qui implante notre réseau de neurones afin de faciliter et d'accélérer les calculs. La précision du modèle est évaluée par l'erreur relative (MRE), l'indicateur Pred (0,25), min des MREs, max des MREs, médiane des MREs et la moyenne des MREs (MMRE).

Nous avons effectué plusieurs essais avec des données empiriques pour choisir le nombre des unités cachées. Les différents essais utilisent la totalité des projets logiciels de la base de données COCOMO'81 pour l'apprentissage et les tests du réseau. Une précision acceptable a été obtenue avec 13 unités cachées et $3\times10^5$ itérations d'apprentissage (Tab. 5.1).

**Tableau 5.1**
Les valeurs des MREs obtenues par un réseau de neurones à 13 unités cachées

| MRE% | Réseau de neurones avec 13 unités cachées |
|---|---|
| Max | 16,67 |
| Moyenne (MMRE) | 1,50 |
| Min | 0,00 |

Cependant, quelques études ont illustré qu'un réseau de neurones fournit des estimations moins satisfaisantes quand les données d'apprentissage sont différentes de celles de tests (Serluca, 1995; Srinivasan et Fisher, 1995). Ceci peut être dû, en particulier, aux trois raisons suivantes:

- Le nouveau projet est totalement différent des projets utilisés dans la phase d'apprentissage.
- Le nombre de projets utilisés dans le processus d'apprentissage est insuffisant.
- Il n'existe pas de relation entre les facteurs du coût considérés (entrée du réseau) et l'effort (sortie du réseau).

Pour confirmer la deuxième observation citée ci-dessus, nous avons réalisé deux essais avec notre réseau de neurones. Dans le premier, nous avons aléatoirement supprimé 23 projets de la base de données COCOMO'81; ces 23 projets seront utilisés pour le test du réseau. Les 40 autres projets seront utilisés pour l'apprentissage. Dans le deuxième essai, nous avons supprimé de la base COCOMO'81 un seul projet qui sera utilisé pour le test du réseau; les 62 projets restants seront utilisés pour l'apprentissage. Le projet supprimé dans le deuxième essai est l'un des 23 projets supprimés dans le premier essai. Le deuxième essai est répété 23 fois avec, à chaque fois, un projet différent. Le tableau 5.2 montre les résultats obtenus de ces deux expériences. Les différents critères d'évaluation des estimations (MRE, Pred(0,25), etc.)

montrent clairement que la précision augmente en fonction du nombre de projets utilisés dans l'apprentissage.

**Tableau 5.2**
Précision des estimations du Perceptron à trois couches en fonction du nombre de projets d'apprentissage et de test

| Nombre de projets: Apprentissage/Test | Min MRE % | Max MRE % | Moy. MRE % | Méd. MRE % | Pred(0,25) % |
|---|---|---|---|---|---|
| 40/23 | 2,6 | 1188,89 | 203,66 | 92,53 | 13,04 |
| 62/1 | 2,84 | 432,5 | 84,35 | 53,67 | 34,78 |

Après cette évaluation empirique de notre réseau de neurones, l'objectif est d'expliquer son fonctionnement en donnant une interprétation compréhensible à la connaissance contenue dans son architecture. Pour ceci, nous utilisons la méthode de Benitez qui transforme un Perceptron de trois couches, comme celui utilisé dans ce travail, en un système à base de règles floues Si-Alors (Benitez, Castro et Requena, 1997).

### 5.4.2 Équivalence entre les réseaux de neurones et les systèmes de règles floues

Les systèmes à base de règles floues Si-Alors sont certainement l'une des contributions majeures de la logique floue pour la résolution de problèmes complexes. Un système à base de règles floues (FRBS) est constitué principalement d'un ensemble de règles floues Si-Alors représentant la connaissance dans le domaine. Une règle floue est une expression Si-Alors dont la prémisse et la conséquence consistent en des propositions floues (Sect. 2.8.1). Un exemple de règles floues en estimation du coût est: *Si la compétence des analystes est élevée Alors l'effort de développement est faible*. L'avantage principal des règles floues par rapport aux règles classiques est qu'elles sont facilement compréhensibles. En effet, les règles floues utilisent des valeurs linguistiques tandis que les règles classiques utilisent des valeurs numériques dans leurs prémisses et leurs conséquences. Par conséquent, plusieurs travaux de recherche ont été entrepris pour établir une équivalence entre les réseaux de neurones et les systèmes à base de règles floues (Jang et Sun, 1992; Buckley, Hayashi et Czogala, 1993; Benitez, Castro et Requena, 1997). Ces travaux ont pour objectif de représenter la connaissance encodée dans le réseau de neurones par un langage compréhensible tel que celui des règles floues Si-Alors. Benitez, Castro et Requena ont développé une méthode qui

transforme un réseau de neurones, comme celui présenté dans la section précédente, en un système à base de règles floues de type Sugeno (Benitez, Castro et Requena, 1997). Dans ce qui suit, nous présentons cette méthode.

Considérons un réseau de neurones de trois couches, avec la fonction Sigmoïde pour les unités cachées et la fonction Identité pour l'unité de sortie. Ce réseau de trois couches est équivalent à un système de règles floues dont les règles $R_j$ sont associées aux paires de ses unités (cachée, sortie).

$$R_j : Si \sum_{i=1}^{n} x_i w_{ij} \; est \; A \quad Alors \; y = b_j \qquad j = 1, \cdots, h$$

où $x_i$ sont les entrées, $y$ est la sortie, $w_{ij}$ sont les poids entre la couche d'entrée et la couche cachée, $b_j$ sont les poids entre la couche cachée et la couche de sortie, et $A$ est un ensemble flou dont la fonction d'appartenance est la fonction Sigmoïde (Équation 1.14). Le nombre de règles floues $R_j$ , $h$, est égal au nombre d'unités de la couche cachée. Afin de rendre les règles floues $R_j$ facilement interprétables, Benitez, Castro et Requena ont démontré que chacune d'elles pourra être exprimée par:

$$R_j : Si \; x_1 \; est \; A_j^1 * x_2 \; est \; A_j^2 * \cdots * x_n \; est \; A_j^n \quad Alors \; y = b_j$$

où $A_j^i$ sont des ensembles flous obtenus de $A$ et $w_{ij}$ . Leurs fonctions d'appartenance sont définies par $\mu_{A_j^i}(x) = \mu_A(x w_{ij})$ et * est l'opérateur *i-or* défini par:

$$i\text{–}or(a_1, \cdots, a_n) = \frac{a_1 \cdots a_n}{(1-a_1)\cdots(1-a_n) + a_1 \cdots a_n} \qquad \text{(Équation 5.2)}$$

Selon Benitez, Castro et Requena, la proposition floue $x \; est \; A_j^i$ est interprétée par *x est approximativement plus grand que* $2{,}2/w_{ij}$ si $w_{ij}$ est positive ou *x n'est pas approximativement plus grand que* $2{,}2/-w_{ij}$ si $w_{ij}$ est négative.

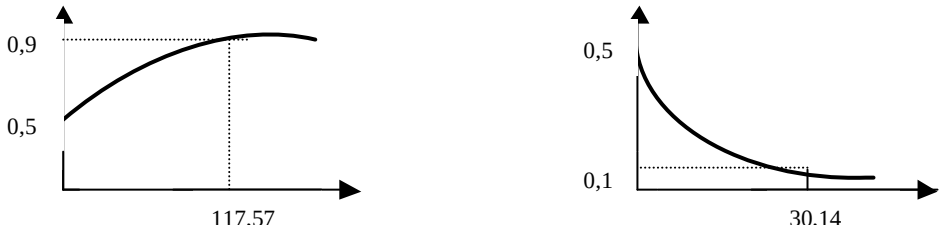
### 5.4.3 Validation et interprétation des règles floues

Dans cette section, nous appliquons la méthode développée par Benitez, Castro et Requena au réseau de neurones présenté et discuté dans la section 5.4.1. Le réseau de neurones considéré a 13 unités cachées et utilise tous les projets de la base de données COCOMO'81 dans le processus d'apprentissage. Par conséquent, la base de règles obtenue contient 13 règles floues. La prémisse de chaque règle floue est composée de 13 propositions floues; chacune est associée à un facteur de coût (entrée du réseau). La conséquence de chaque règle est une valeur numérique (positive ou négative). Ces 13 règles floues représentent la connaissance encodée par les poids synaptiques du réseau. L'objectif est de fournir une interprétation compréhensible de ces règles floues en utilisant notre représentation floue des 12 attributs du modèle COCOMO'81 (Idri, Abran et Kjiri, 2000). Pour fins d'illustration et de simplification, notre discussion se limitera à deux règles floues (Tab. 5.3).

En analysant ces deux règles, nous remarquons que la sortie de la première règle, $R_1$, est positive (6049,77) alors que celle de la deuxième, $R_2$, est négative (-2979,21). Ces deux valeurs représentent les poids synaptiques entre la couche cachée et la couche de sortie. Ainsi, l'interprétation naturelle que nous pouvons donner aux sorties de ces deux règles est qu'elles représentent des contributions partielles à l'effort total de développement. Elles ont la même signification que les multiplicateurs d'effort du modèle COCOMO'81. Elles peuvent augmenter (valeur positive) ou diminuer (valeur négative) l'effort total. La seule différence entre les deux est que les sorties des règles floues sont des valeurs positives ou négatives, car la fonction coût du système flou utilise l'opérateur Sommation, alors que les multiplicateurs d'effort dans COCOMO'81 sont supérieurs à 1 (augmentent l'effort) ou inférieurs à 1 (diminuent l'effort) car la fonction coût du modèle COCOMO'81 utilise l'opérateur Multiplicateur.

La prémisse de chaque règle floue est composée de 13 propositions floues combinées par l'opérateur *i-or*. Chaque proposition est traduite par l'expression *x est approximativement plus grand que v* ou *x n'est pas approximativement plus grand que v*. La valeur linguistique *approximativement plus grand que v* est représentée par un ensemble flou dont la fonction d'appartenance est la fonction Sigmoïde. La valeur *v* est celle dont le degré d'appartenance

est égal à 0,9 dans le cas *approximativement plus grand que v* ou 0,1 dans le cas *n'est pas approximativement plus grand que v*. Dans la littérature des réseaux de neurones, les valeurs 0,1 et 0,9 sont utilisées pour indiquer respectivement l'absence et la présence totale de l'activation des neurones (Benitez, Catro et Requena, 1997). Dans notre application, les entrées du réseau ne peuvent avoir que des valeurs positives; par conséquent, nous utilisons seulement la partie positive des domaines des ensembles flous correspondant aux deux valeurs linguistiques *est approximativement plus grand que v* et *n'est pas approximativement plus grand que v*. La figure 5.3 montre deux exemples qui illustrent les ensembles flous des deux premières propositions de la règle $R_1$.




(a) (b)

**Figure 5.3** (a) Ensemble flou associé à la qualification *approximativement plus grand que 117,57*. (b) Ensemble flou associé à la qualification *n'est pas approximativement plus grand que 30,14*.

**Tableau 5.3**

Exemples de deux règles floues obtenues du Perceptron à trois couches

| $R_1$ | PRÉMISSE | | |
|---|---|---|---|
| I-OR | DATA | ***est*** ~>[9] | 117,57 |
| | VIRTmin | ***n'est pas*** ~> | 30,14 |
| | TIME | ***n'est pas*** ~> | 156,28 |
| | STORE | ***n'est pas*** ~> | 155,72 |
| | VIRTmaj | ***n'est pas*** ~> | 448,88 |
| | TURN | ***n'est pas*** ~> | 25,76 |
| | ACAP | ***n'est pas*** ~> | 203,72 |
| | AEXP | ***n'est pas*** ~> | 400,37 |
| | PCAP | ***n'est pas*** ~> | 331,69 |
| | VEXP | ***n'est pas*** ~> | 1958,86 |
| | LEXP | ***n'est pas*** ~> | 921,91 |
| | SCED | ***n'est pas*** ~> | 794,68 |
| | AKDSI | ***n'est pas*** ~> | 132,27 |
| | CONSÉQUENCE | | |
| | Y = 6049,77 | | |

| $R_2$ | PRÉMISSE | | |
|---|---|---|---|
| I-OR | DATA | ***est*** ~> | 118,46 |
| | VIRTmin | ***n'est pas*** ~> | 18,14 |
| | TIME | ***est*** ~> | 3716,39 |
| | STORE | ***n'est pas*** ~> | 573,41 |
| | VIRTmaj | ***est*** ~> | 1232,53 |
| | TURN | ***n'est pas*** ~> | 46,33 |
| | ACAP | ***n'est pas*** ~> | 393,35 |
| | AEXP | ***n'est pas*** ~> | 272,94 |
| | PCAP | ***n'est pas*** ~> | 483,23 |
| | VEXP | ***est*** ~> | 1357,35 |
| | LEXP | ***est*** ~> | 985,36 |
| | SCED | ***n'est pas*** ~> | 171,20 |
| | AKDSI | ***est*** ~> | 176,79 |
| | CONSÉQUENCE | | |
| | Y = -2979,21 | | |

[9] ~> signifie *approximativement plus grand que*

L'interprétation de chaque proposition floue dépend de la signification du facteur coût qui lui est associée. Par exemple, la proposition *DATA est approximativement plus grand que 117,57* utilise le facteur DATA évalué dans COCOMO'81 par le ratio suivant:

$$\frac{D}{P} = \frac{Taille\ de\ la\ base\ de\ données}{Taille\ du\ programme\ en\ ISL}$$

l'attribut DATA représente l'effet de la taille de la base de données sur l'effort de développement de logiciels. Ainsi, plus la taille de la base de données est grande plus elle influence positivement l'effort total. En utilisant notre représentation floue de l'attribut *DATA* (Fig. 3.3), nous remarquons que la valeur 117,57 appartient à la valeur linguistique *élevé* ou *très élevé*. Par conséquent, la proposition floue *DATA est approximativement plus grand que 117,57* peut être considérée comme équivalente à *DATA est élevé* ou *très élevé*. Cette dernière a l'avantage d'être facilement comprise en estimation des coûts (Fig. 3.3).

Cependant, dans plusieurs cas la valeur *v* n'appartient pas à l'intervalle des valeurs permises pour un attribut particulier. Par exemple, dans la règle $R_1$, la proposition *LEXP n'est pas approximativement plus grand que 921,91* utilise la valeur 921,91; cette valeur n'appartient pas à l'intervalle des valeurs possibles de l'attribut LEXP (Fig. 5.4). En effet, l'attribut LEXP représente l'expérience des programmeurs et il est mesuré par le nombre de mois d'expérience. Dans le modèle COCOMO'81, la plus grande valeur possible est 36 mois. En utilisant notre représentation floue de l'attribut LEXP (Annexe A), la proposition *LEXP n'est pas approximativement plus grand que 921,91* peut être considérée comme à *LEXP est Q(very low)* où *Q* est un modificateur linguistique tel que *plus que*.

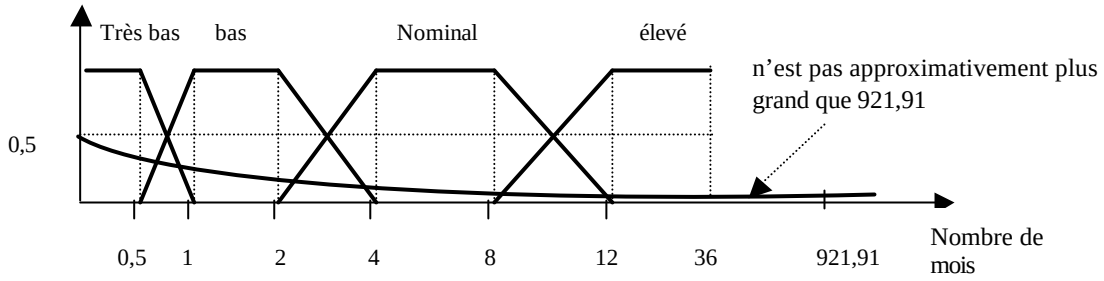



**Figure 5.4** Exemple où la valeur utilisée pour le facteur LEXP n'est pas dans l'intervalle permis.

Dans ce qui précède, nous avons proposé une interprétation des prémisses et des conséquences des règles floues. Il reste à expliquer la signification de l'opérateur *i-or*. Selon Benitez, Castro et Requena, l'opérateur *i-or* a une interprétation naturelle et peut être utilisé dans l'évaluation de plusieurs situations du monde réel telles que l'évaluation des articles scientifiques et l'évaluation de la qualité du jeu de deux joueurs de tennis. En analysant les 13 règles floues, il nous semble que l'opérateur *i-or* n'est pas approprié pour évaluer l'effet des prémisses sur l'effort. En effet, l'influence d'une prémisse sur l'effort est évaluée en considérant toutes les propositions floues composant la prémisse. Chaque proposition floue est associée à un facteur du coût. En estimation du coût, l'effet d'un facteur sur l'effort dépend de son type et de son importance. En plus l'opérateur *i-or* ne modélise pas convenablement les relations complexes existantes entre les facteurs du coût. Par exemple, supposons que la valeur de vérité d'une proposition est égale à 1 et que toutes les autres propositions de la prémisse ont des valeurs de vérité au voisinage de 0; la combinaison de ces propositions par *i-or* donne une valeur de vérité égale à 1. Ceci est en contradiction avec notre intuition, spécifiquement si la proposition qui a une valeur de vérité égale à 1 est associée au facteur le moins significatif pour l'effort.

En conclusion de cette étude préalable que nous avons menée pour interpréter un Perceptron à trois couches en estimation des coûts, nous avons pu fournir une interprétation compréhensible aux propositions floues composant la prémisse ainsi qu'à la conséquence de chaque règle floue. Cependant, la combinaison des différentes propositions floues par l'opérateur *i-or* semble inappropriée en estimation des coûts. De plus, l'opérateur *i-or* ne peut être facilement expliqué dans plusieurs situations où il génère des sorties contradictoires à notre intuition. Par conséquent, nous abandonnons l'idée d'établir une équivalence entre le

Perceptron multicouche et notre approche *Fuzzy Analogy* par l'intermédiaire d'un système à base de règles floues. Ainsi, nous développons, dans les deux sections suivantes, un cadre théorique pour établir une équivalence directe entre les réseaux de neurones, spécifiquement un réseau RBFN, et notre approche *Fuzzy Analogy*.

## 5.5 ÉQUIVALENCE ENTRE LES RÉSEAUX DE NEURONES ET *FUZZY ANALOGY*: CAS DU RÉSEAU RBFN

Comme nous l'avons déjà mentionné dans la section 4.1, le raisonnement par analogie généralise plusieurs types de réseaux de neurones. En particulier, nous avons cité le cas du Perceptron simple, le réseau de Kohonen, les réseaux RBFN (*Radial Basis Functions Network*) et le réseau de Hopfield. Nous présentons ici, brièvement, les cas du Perceptron simple, le réseau de Kohonen et le réseau de Hopfiled. Cependant, le cas du réseau RBFN fera l'objet d'une étude détaillée que nous présenterons dans les deux sous-sections 5.5.1 et 5.5.2.

Dans un Perceptron simple (deux couches: une couche d'entrée et une couche de sortie), la sortie $Y_i$ est déterminée par la formule suivante:

$$Y_i = g(\sum_k w_{ik} x_k) \qquad \text{(Équation 5.3)}$$

où $x_k$ sont les entrées du réseau, $w_{ik}$ sont les poids de connexions et $g$ représente la fonction d'activation des neurones de sorties, $Y_i$. Si nous considérons que $g$ est la fonction Identité, l'équation 5.3 devient:

$$Y_i = \sum_k w_{ik} x_k \qquad \text{(Équation 5.4)}$$

L'équation 5.4 représente, en fait, le produit scalaire entre le vecteur des poids, $w$, et celui des entrées $x$, $<w.x>$. Or, nous savons que le produit scalaire entre $w$ et $x$ est donné par l'équation suivante:

$$< w.x > = \|w\| \|x\| \cos(w, x)$$

où $\|.\|$ représente la Norme d'un vecteur et $cos(w,x)$ est le Cosinus de l'angle formé par $w$ et $x$. Si nous supposons que les Normes des vecteurs $w$ et $x$ sont toutes les deux égales à un, le

produit scalaire de $w$ et $x$ serait égal au Cosinus de l'Angle formé par $w$ et $x$. Ce Cosinus représente le degré de similarité entre $w$ et $x$: s'il est au voisinage de 1, cela implique que l'angle formé par $w$ et $x$ est au voisinage de 0, c'est-à-dire, les deux vecteurs sont liés (similaires); s'il est au voisinage de $-1$, cela implique que l'angle formé par $w$ et $x$ est au voisinage de 180°, c'est-à-dire que les deux vecteurs sont opposés (non similaires; Figure 5.5). En résumé, un Perceptron simple ne fait donc que calculer la similarité entre deux vecteurs: celui représentant l'entrée avec celui représentant les poids de la couche d'entrée à la couche de sortie.

Le réseau de Kohonen est inspiré de l'organisation du cortex cérébral où des zones qui sont physiquement proches (similaires) dans le cortex visuel correspondent aussi à des zones proches dans la rétine (Hertz, Krogh et Palmer, 1991; Hayken, 1994). Par conséquent, son principe est de préserver les relations de similarité existantes entre ses entrées en les retrouvant aussi au niveau de ses sorties. Nous retrouvons donc bien l'affirmation du raisonnement par analogie: *similar inputs have similar outputs.* Le réseau de Kohonen est un réseau à deux couches: la première couche représente la couche d'entrée et la deuxième couche représente les différentes catégories où les patrons (entrées) seront classifiés (couche compétitive). Chaque neurone de la couche de sortie désigne une catégorie. Kohonen adopte une approche computationnelle pour déterminer le neurone gagnant de la couche de sortie. En effet, le neurone gagnant est celui qui satisfait l'inégalité suivante:

$$< w_{i^*}.x > \geq < w_i.x > \quad \forall i$$

où $w_i$ est le vecteur des poids du neurone de sortie, i, et $x$ est le vecteur représentant le patron à classifier. Le neurone gagnant n'est donc en fin de compte que celui ayant le vecteur $w_i$ le plus similaire à $x$.

Le réseau de Hopfield est composé de neurones entièrement connectés les uns aux autres. Les connexions sont directes et dirigées dans les deux sens pour toute paire de neurones. Initialement, Hopfield avait proposé son réseau pour l'implantation des mémoires associatives. Ainsi, le réseau mémorise un certain ensemble de connaissances et peut donc

reconnaître une connaissance bruitée en cherchant celle qui est la plus similaire parmi toutes celles déjà mémorisées. Le réseau de Hopfield réalise donc un raisonnement par analogie.

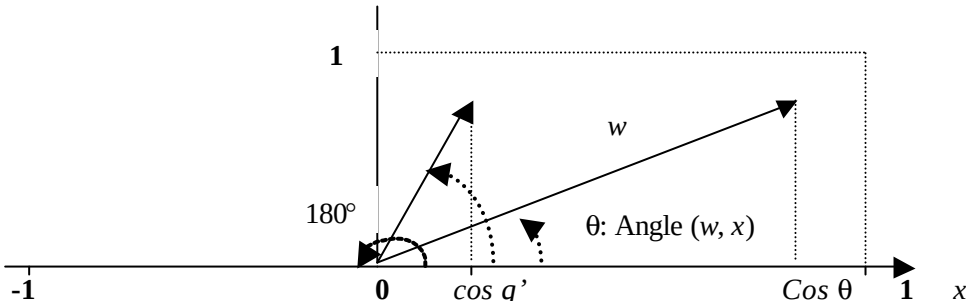



**Figure 5.5** Illustration de la relation entre le produit scalaire et la similarité entre deux vecteurs *x* et *w*.

5.5.1 *Fuzzy Analogy* vs réseau RBFN: un cadre théorique pour l'établissement d'une Équivalence

Les réseaux RBFN ont été développés par Moody et Darken (1989). Ils ont été utilisés avec succès dans plusieurs domaines du fait qu'ils peuvent se rapprocher de plusieurs types de fonctions (Park et Sandberg, 1993). Un réseau RBFN est composé de trois couches: une couche d'entrée, une couche cachée et une couche de sortie. Chaque neurone de la couche d'entrée est connecté à tous les neurones de la couche cachée (Fig. 5.6). La fonction d'activation des neurones cachés est, en général, la fonction gaussienne (Fig. 5.7):

$$f(x) = e^{\left(-\frac{\|x-c_i\|}{2s_i^2}\right)} \qquad \text{(Équation 5.5)}$$

où $c_i$ et $s_i$ sont respectivement le centre et le rayon de la fonction gaussienne et $\|.\|$ dénote la distance euclidienne entre $x$ et $c_i$. Les poids de la couche d'entrée au $i^{ème}$ neurone de la couche cachée sont les composants du centre $c_i$. La fonction d'activation des neurones de sortie est la fonction Identité. Ainsi, la sortie $Z$ du réseau RBFN est déterminée par:

$$Z = \sum_{i=1}^{M} b_j y_j$$

$$y_j = e^{\left(\frac{\|x - c_j\|}{2s_j^2}\right)}$$

Les réseaux RBFN utilisent un apprentissage hybride pour la détermination des paramètres $c_i$, $s_i$ et $b_j$ (Moody et Darken, 1989; Hertz, Krogh et Palmer, 1991). Ils utilisent un apprentissage compétitif (non supervisé) pour $c_i$ et $s_i$, et un apprentissage supervisé (descende du gradient) pour les $b_j$.

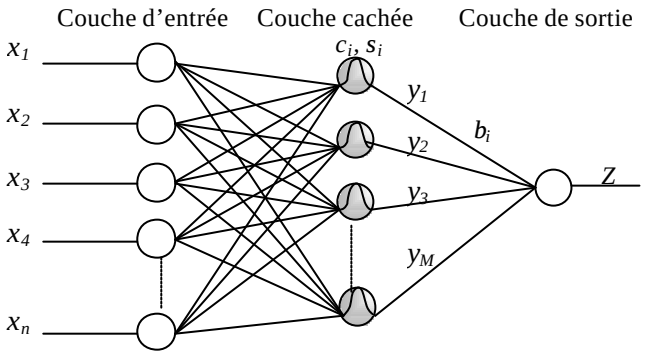



**Figure 5.6** Exemple d'un réseau RBFN avec un seul neurone dans la couche de sortie.

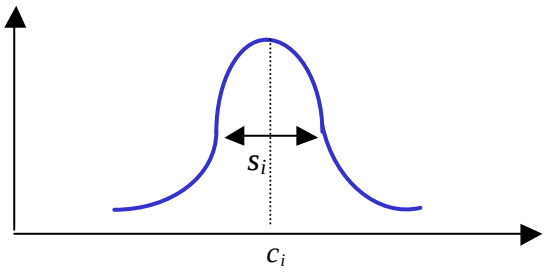



**Figure 5.7** Fonction gaussienne.

L'équivalence entre *Fuzzy Analogy* et un réseau RBFN s'inspire de l'architecture et de l'algorithme d'apprentissage adoptés par un réseau RBFN. En effet, *Fuzzy Analogy* est composé de trois étapes qui peuvent être réalisées par les trois couches du réseau RBFN. La première couche correspondra à l'étape d'Identification des projets logiciels. Ainsi, les

neurones de la couche d'entrée, $x_i$, représenteront respectivement les différents attributs décrivant un projet logiciel. Chaque neurone, i, de la couche d'entrée recevra la valeur numérique correspondante à l'attribut $x_i$ du projet logiciel. La deuxième couche correspondra à l'étape d'Évaluation de la similarité entre les projets logiciels. En effet, à chaque neurone de la couche cachée correspond un centre $c_j$ et un rayon $s_j$ qui définissent sa fonction d'activation. Cette fonction d'activation, la fonction gaussienne, détermine, en fait, le degré de similarité entre le vecteur d'entré $x(x_1, x_2,..,x_n)$ représentant le nouveau projet logiciel P et le centre $c_j$. Ainsi, si nous considérons que les centres $c_j$ sont respectivement les projets logiciels historiques $P_j$, chaque neurone, j, de la couche cachée déterminera donc la similarité entre le nouveau projet P et le $j^{ème}$ projet historique (centre $c_j$) de la base de projets, $d(P, P_j)$. Par conséquent, nous aurons autant de neurones dans la couche cachée que de projets logiciels historiques. Cependant, cette solution présente l'inconvénient de générer un nombre élevé de neurones dans la couche cachée si le nombre de projets logiciels historiques est aussi élevé.

Pour y remédier, nous utilisons des algorithmes de classification pour regrouper les projets logiciels historiques similaires dans les mêmes classes. En effet, les réseaux RBFN utilisent souvent ces algorithmes de classification (*clustering algorithms*) tels que le *K-means* ou le *Self-organizing* pour déterminer le nombre de classes (*clusters*) existantes dans la population initiale de patrons et affecter ensuite, à chaque classe un neurone dont $c_j$ est le centre de la classe. Les algorithmes de classification adoptent, en général, un apprentissage non supervisé et compétitif pour la détermination de ces centres. En effet, les patrons sont présentés plusieurs fois de la couche d'entrée à la couche cachée et pour chaque présentation d'un patron, l'algorithme détermine le neurone gagnant. C'est ce neurone qui représentera la classe du patron. L'algorithme assure que deux patrons similaires appartiendront à la même classe; autrement dit, ils activeront le même neurone de la couche cachée. Le centre d'une classe, $c_j$, est déterminé en fonction des patrons appartenant à la classe. Nous présentons dans la section suivante un exemple d'un algorithme de classification qui servira pour valider empiriquement un réseau RBFN sur les projets COCOMO'81.

Ainsi, chaque neurone de la couche cachée du réseau RBFN, équivalent à *Fuzzy Analogy*, calcule la similarité entre le nouveau projet, P, et le projet représentant le centre de la classe associée au neurone, $c_j$. La similarité entre P et $c_j$ est donc déterminée par:

$$d(P, c_j) = e^{\left(-\frac{\|P - c_j\|}{2s_j^2}\right)} \qquad \text{(Équation 5.6)}$$

L'équation 5.6 peut s'écrire aussi comme:

$$d(P, c_j) = \prod_{i=1}^{M} e^{\left(-\frac{(P(x_i) - c_j(x_i))^2}{2s_j^2}\right)} = \prod_{i=1}^{M} d_{x_i}(P, c_j) \quad \text{(a)}$$

$$avec \quad d_{x_i}(P, c_j) = e^{\left(-\frac{(P(x_i) - c_j(x_i))^2}{2s_j^2}\right)} \quad \text{(b)} \qquad \text{(Équation 5.7)}$$

où $x_i$ sont les attributs décrivant les projets logiciels et M est le nombre de ces attributs (nombre de neurones de la couche d'entrée). $d_{x_i}(P, c_j)$ représente la similarité individuelle entre $P$ et $c_j$ selon l'attribut $x_i$. Les mesures de similarité d'un réseau RBFN adoptent donc la même stratégie que celles utilisées dans *Fuzzy Analogy*, c'est-à-dire qu'elles évaluent la similarité globale en combinant les similarités individuelles. Cependant, les définitions de ces mesures ne sont pas les mêmes que celles adoptées dans la cas de *Fuzzy Analogy*. En effet,

- *Fuzzy Analogy* adopte des mesures de similarité individuelle qui utilisent les techniques d'agrégation floues (*max-min* et *sum-product*, Équations 4.3 et 4.4) alors que celles d'un réseau RBFN utilisent la fonction gaussienne (Équation 5.7b).

- *Fuzzy Analogy* utilise les quantificateurs linguistiques RIM pour évaluer la similarité globale en fonction des similarités individuelles (Équation 4.6) alors que le réseau RBFN utilise l'opérateur Produit $P$ (Équation 5.7a).

Concernant la similarité individuelle, nos mesures supposent que les valeurs linguistiques, ainsi que leurs représentations floues de chaque attribut $x_i$ sont déterminées d'avance (voir Annexe A pour le cas des attributs du modèle COCOMO'81); par contraste, celles du réseau RBFN sont déterminées à l'aide d'un apprentissage compétitif afin de faire converger le

réseau. En effet, chaque centre $c_j$ d'un neurone de la couche cachée fournit une valeur linguistique, ainsi que sa représentation floue par la fonction gaussienne de l'attribut $x_i$. Le nombre de valeurs linguistiques de chaque attribut est donc égal au nombre de neurones de la couche cachée. Nous incorporons ces mesures de similarité individuelle dans *Fuzzy Analogy* afin de lui permettre la mise à jour automatique de tous ses paramètres en relation avec les valeurs linguistiques qui décrivent les projets logiciels (nombre de valeurs linguistiques, définitions de leurs ensembles floues, formes des fonctions d'appartenance aux ensembles flous, etc.).

En ce qui a trait à la similarité globale, nos mesures utilisent des quantificateurs linguistiques RIM tels que *most*, *many* et *at-least α* pour combiner les différentes similarités individuelles. Un réseau RBFN utilise, en général, l'opérateur Produit. Comme nous l'avons souligné dans la section 4.4.2, l'utilisation d'un quantificateur linguistique offre deux avantages:

- Il permet la prise en considération des poids associés aux attributs $x_i$.
- Il permet le choix de la combinaison adéquate dépendamment de l'environnement étudié.

Quant à l'opérateur Produit utilisé dans un RBFN, en plus de ne pas permettre une flexibilité dans l'évaluation de la similarité globale, il pourra mener, dans plusieurs cas, à des situations contradictoires à notre intuition. Par exemple, si deux similarités individuelles sont évaluées toutes les deux à 0,5, leur combinaison par l'opérateur Produit fournit une similarité globale égale à 0,25. De ce fait, nous maintenons, dans *Fuzzy Analogy*, l'utilisation d'un quantificateur linguistique pour l'évaluation de la similarité globale. Cependant, nous adoptons, pour les neurones de la couche cachée du réseau RBFN, une fonction d'activation identique à la fonction d'appartenance qui définit le quantificateur linguistique utilisé dans *Fuzzy Analogy*. Nous aurons donc une équivalence totale entre la couche cachée du réseau RBFN et l'étape d'Évaluation de la similarité du *Fuzzy Analogy*:

$$d(P,c_j) = \begin{cases} all\ of\ (d_{x_i}(P,c_j)) \\ most\ of\ (d_{x_i}(P,c_j)) \\ many\ of\ (d_{x_i}(P,c_j)) \\ .... \\ there\ exists\ of\ (d_{x_i}(P,c_j)) \end{cases} \qquad \text{(Équation 5.8)}$$

où $d_{x_i}(P,c_j)$ est donnée par l'équation 5.7 (b).

L'étape d'Adaptation du *Fuzzy Analogy* correspondra à la couche de sortie du réseau RBFN. Nous considérons qu'il y a un seul neurone dans la couche de sortie. Ce neurone fournira le coût estimé du nouveau projet logiciel:

$$cout = \sum_{j=1}^{N} b_j d(P,c_j) \qquad \text{(Équation 5.9)}$$

où $b_j$ sont les poids de la couche cachée au neurone de sortie. Pour avoir une équivalence partielle entre l'équation 5.9 et celle représentant l'étape d'Adaptation du *Fuzzy Analogy* (Équation 4.9), nous pourrons choisir $b_j$ = coût($c_j$). Ainsi, le coût d'un nouveau projet logiciel P, qui est similaire seulement à un seul centre $c_j$, serait exactement égal au coût du $c_j$, si P est similaire à deux centres $c_j$ et $c_{j'}$, son coût serait égal à la moyenne pondérée des coûts de $c_j$ et $c_{j'}$. Cependant, dans un réseau RBFN, les poids $b_j$ sont, en général, déterminés par un apprentissage supervisé utilisant par exemple la règle Delta. Dans notre cas, la règle Delta est formulée par:

$$\Delta b_j = h(cout_{est}(P) - cout_{réel}(P))d(P,c_j) \qquad \text{(Équation 5.10)}$$

où $\eta$ est le taux d'apprentissage (souvent entre 0 et1), $cout_{est}(P)$ est le coût estimé de P par le réseau RBFN, et $cout_{réel}(P)$ est le coût réel de P. Dans notre réseau RBFN équivalent à *Fuzzy Analogy*, nous adoptons la règle Delta de l'équation 5.10 pour retrouver les $b_j$ qui permettent au réseau RBFN de fournir les meilleures estimations des coûts. Aussi, cette formule réalise implicitement le concept de la valeur linguistique *étroitement similaire* que nous avons utilisé dans l'étape d'Adaptation de *Fuzzy Analogy*. La figure 5.8 présente l'architecture générale du réseau RBFN équivalent à notre technique *Fuzzy Analogy*.

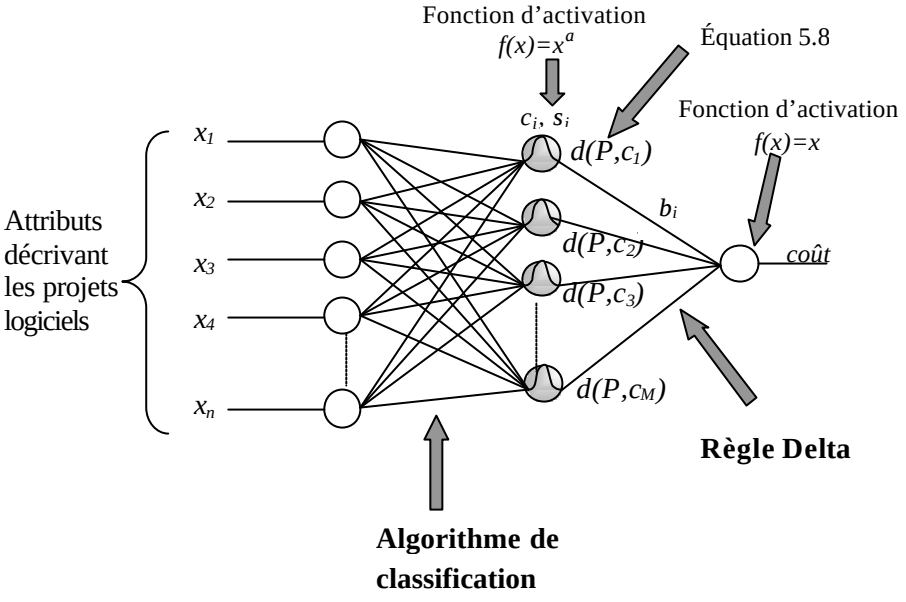



**Figure 5.8** Architecture du réseau RBFN équivalent à *Fuzzy Analogy*.

5.5.2 Validation du réseau RBFN sur les projets du COCOMO'81

Dans la section précédente, nous avons présenté l'architecture générale du réseau RBFN équivalent à *Fuzzy Analogy* (Fig. 5.8). La configuration de cette architecture nécessite essentiellement la détermination des centres de la couche cachée. Pour ce faire, nous utilisons l'algorithme de classification APC-III développé par Hwang et Bang (1997). APC-III est un algorithme du type *one-pass*, c'est-à-dire qu'il détermine en une seule itération les différents *clusters* à partir d'une population de patrons. Ainsi, il calcule tout d'abord un rayon $R_0$ qui est proportionnel à la moyenne des distances minimales entre les patrons de la population initiale:

$$R_0 = l \frac{1}{N} \sum_{i=1}^{N} \min_{i \neq j} \left( \left\| X_i - X_j \right\| \right) \qquad \text{(Équation 5.11)}$$

où N est le nombre de patrons de la population initiale et $\lambda$ est une constante à choisir. $R_0$ exprime le rayon de chaque *cluster*. Par conséquent, si $R_0$ est assez petit, le nombre de

*clusters* serait élevé et vice versa. $R_0$ contrôle donc le nombre de *clusters* déterminés par l'algorithme APC-III. Dans une deuxième étape, APC-III cherche, pour chaque patron $X_i$, le premier *cluster* $C_j$ dont le centre $c_j$ a une distance euclidienne avec $X_i$ inférieure ou égale à $R_0$. L'ajout de $X_i$ à $C_j$ modifie le centre $c_j$. Si $X_i$ n'appartient à aucun *cluster*, un nouveau *cluster* est créé dont le centre est $X_i$. La figure 5.9 décrit l'algorithme APC-III.

```
Entrée:  vecteur de patrons X=(x1,x2,…xP)
Sortie:  centres de clusters cj
Variables:
   C:  Nombre de clusters
   cj: centre du jème cluster
   nj: number de patrons dans le jème cluster
   dij distance entre xi et le jème cluster
C=1; c1= x1; n1=1;
Pour i=2 à P faire        /* pour chaqur patron*/
    Pour j=1 à C    /* pour chaque cluster*/
      Calculer dij
     Si dij ≤R0 alors
    /* ajouter xi dans le jème cluster*/
       cj =( cj  nj  +xi)/ ni +1
       ni = ni +1
       sortir de la boucle
     fin Si
  fin Pour
  Si xi n'est pas ajouté à aucun cluster  /* créer un nouveau cluster */
  C=C+1
  cC= xi
  nc=1
  Fin Si
Fin Pour
```

**Figure 5.9** Algorithme APC-III (Hwang et Bang, 1997).

Nous appliquons l'algorithme APC-III sur les 63 projets du COCOMO'81. Chaque projet est décrit par 13 attributs: la taille du logiciel et les 12 attributs que nous avons déjà utilisé dans la validation de *Fuzzy Analogy* (Annexe A). Par conséquent, le réseau RBFN aura 13 neurones dans la couche d'entrée ($x_i$). Le nombre de neurones de la couche cachée est évalué par le nombre de *clusters* générés par l'algorithme APC-III. Pour ce faire, il faut choisir une valeur de $\lambda$ afin de déterminer le rayon $R_0$. Le tableau 5.4 résume les résultats obtenus du nombre de *clusters* en fonction de la valeur $\lambda$. Nous remarquons que le nombre de *clusters* décroît en fonction de $\lambda$. Ceci est dû au fait que $R_0$ est monotone croissante en fonction de $\lambda$. Pour chaque valeur de $\lambda$, nous avons varié l'ordre de présentation des 63 projets du

COCOMO'81 dans l'algorithme APC-III. En effet, les *clusters* fournis par l'algorithme APC-III dépendent de l'ordre de présentation des projets puisque celui-ci influence les déplacements des centres de ces *clusters*. Par exemple, dans le cas où $\lambda$ est égal à 0,47, le nombre de *clusters* est soit 59 soit 58 du fait que le regroupement ($P_{60}$, $P_{59}$, $P_{27}$ et $P_{50}$) peut être scindé en deux sous-regroupements ($P_{60}$, $P_{59}$) et ($P_{27}$,$P_{50}$), et ceci dépendamment de l'ordre de présentation des projets $P_{60}$, $P_{59}$, $P_{27}$ et $P_{50}$. Aussi, l'influence de l'ordre de présentation des projets sur la classification générée par APC-III devient plus importante quand $\lambda$ est grande (cas de $\lambda$ supérieur à 0,66 dans le tableau 5.4).

**Tableau 5.4**
Nombre de *clusters* en fonction de $\lambda$: APCIII appliqué sur les projets COCOMO'81

| $\lambda$ | Nombre de clusters |
|---|---|
| 0,4/0,45/0,46/0,47/0,49/0,5/ | 62/61 59/60/59 58/56 55/55 54/ |
| 0,52/0,6/0,62/0,66 | 52/51 50/49 48 50/49 48 47 |
| 0,70/0,75/0,78/0,80 | /45 47 48/43 45 46/42 39 41/39 38 37 40 |

Parmi les classifications du tableau 5.4, choisir la meilleure, à utiliser dans le réseau RBFN, n'est pas souvent une tâche facile. En effet, dans notre cas, ce choix doit satisfaire deux objectifs:

1- La classification adoptée doit permettre au réseau RBFN de fournir des estimations acceptables aux coûts des nouveaux projets logiciels, et

2- la classification doit être cohérente, c'est-à-dire les projets d'un *cluster* donné sont nécessairement similaires.

Pour ce faire, nous avons mené plusieurs expérimentations avec un réseau RBFN utilisant, à chaque fois, l'une des classifications du tableau 5.4. Les poids de la couche cachée au neurone de sortie, $b_j$ sont mis à jour selon la règle Delta avec un taux d'apprentissage $\eta$ égal à 0,03. Le réseau RBFN converge rapidement avec un nombre d'itérations d'apprentissage inférieur ou égal à 12 000 et une erreur maximale de l'ordre de $10^{-3}$. La précision des estimations du réseau RBFN est évaluée par les deux indicateurs MMRE et Pred(25). La

Figure 5.10 montre les variations de ces deux indicateurs en fonction de la classification adoptée par le réseau RBFN ($\lambda$). Nous constatons que la précision des estimations du réseau RBFN est meilleure quand $\lambda$ est inférieur ou égal à 0,49 (MMRE $\leq$30 et pred(25)$\geq$70). Par contraste, quand $\lambda$ est supérieur à 0,49, la MMRE devient grande bien que les valeurs de l'indicateur Pred(25) soient satisfaisantes. En effet, la MMRE est très sensible aux variations des estimations. Ainsi, quand la valeur de $\lambda$ augmente, la classification générée par l'algorithme APC-III est moins cohérente, c'est-à-dire que certains regroupements de la classification contenant des projets logiciels ne représentent pas réellement des ressemblances satisfaisantes. Pour ces projets logiciels, le réseau RBFN pourra commettre des imprécisions flagrantes au niveau des estimations de leurs coûts. Afin de vérifier cette hypothèse, nous avons analysé les *clusters* fournis par APC-III en utilisant nos mesures de similarité des équations 4.3, 4.4 et 4.6. Nous avons remarqué que, à partir de $\lambda$ supérieur à 0,49, certains *clusters* contiennent des projets qui ont des degrés de similarité faibles. Par exemple, pour $\lambda$ est égal à 0,5, un nouveau *cluster* est formé et composé des projets $P_{15}$ et $P_{13}$ auxquels s'ajoute le projet $P_{57}$ quand $\lambda$ est égal à 0,66. L'évaluation de la similarité entre ($P_{15}$, $P_{13}$), d'une part, et de celle entre ($P_{13}$, $P_{57}$), d'autre part, indique que, dans le premier cas, les deux projets $P_{15}$ et $P_{13}$ ont six similarités individuelles égales à zéro parmi douze alors que, dans le deuxième cas, les deux projets $P_{13}$ et $P_{57}$ en ont sept. Les MRE respectives aux projets $P_{15}$ et $P_{57}$ sont nettement grandes 100,50% et 118,64%.

La précision des estimations d'un réseau RBFN ne dépend pas seulement du nombre de neurones de la couche cachée, c'est-à-dire du nombre de *clusters* fournis par l'APC-III ($\lambda$); mais aussi de la variance $s_i$ utilisée par la fonction d'activation des neurones de la couche cachée (Équation 5.5). Dans nos expérimentations précédentes, nous avons utilisé une valeur $s_i$ égale à 30 pour tous les neurones de la couche cachée. Cependant, dans la littérature, la valeur $s_i$ est choisie de telle façon à recouvrir uniformément, autant que possible, tout l'espace de patrons (Hertz, Krogh et Palmer, 1991; Hwang et Bang, 1997; Hayken, 1994). Par exemple, si les distances entre les centres de *clusters* ne sont pas uniformes, on préfère choisir ceux très éloignés les uns des autres des valeurs $s_i$ assez grandes et ceux très proches les uns des autres des valeurs $s_i$ assez petites. Dans notre cas, cette stratégie ne nous semble pas appropriée. En effet,

- Un recouvrement uniforme de tout l'espace de patrons implique que le réseau RBFN pourra générer une estimation au coût d'un nouveau projet même s'il ne présente aucune similarité satisfaisante avec les projets logiciels historiques.

- Le choix des valeurs $s_i$ pour recouvrir uniformément tout l'espace de patrons n'est pas unique.

- Dans le cas de la base de projets COCOMO'81, les centres des *clusters*, générés par l'algorithme APC-III, sont souvent très éloignés les uns des autres. Ainsi, les valeurs de $s_i$ doivent être assez grandes pour assurer un recouvrement uniforme de tous les projets du COCOMO'81. Cependant, comme nous l'illustrons ci-dessus, des valeurs assez grandes de $s_i$ mènent souvent à des estimations imprécises dans le cas des projets COCOMO'81.

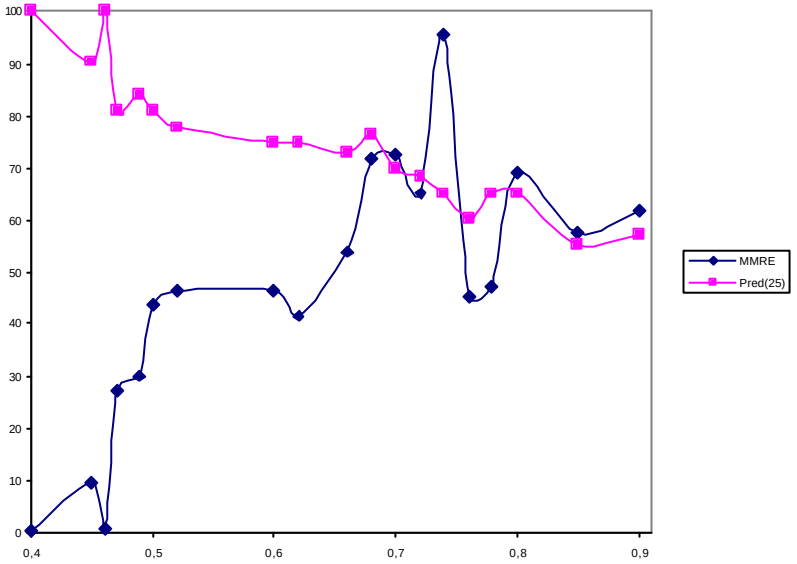



**Figure 5.10** MMRE et Pred(25) du réseau RBFN en fonction de λ.

La stratégie que nous adoptons pour le choix des valeurs $s_i$ se base essentiellement sur deux hypothèses: premièrement, une seule valeur est assignée à toutes les $s_i$; deuxièmement, cette valeur dépendra du rayon $R_0$ de l'algorithme APC-III. En effet, $R_0$ représente la moyenne des

distances minimales entre les projets logiciels d'apprentissage multipliée par le paramètre λ (Équation 5.11). Elle est utilisée dans APC-III pour la délimitation des *clusters*. Il est donc raisonnable que dans les neurones de la couche cachée, les similarités entre les centres et les nouveaux projets logiciels soient évaluées en utilisant une valeur $s_i$ assez proche de $R_0$. Pour valider notre proposition, nous évaluons la précision d'un réseau RBFN en utilisant différentes valeurs de $s_i$ pour la même classification (même valeur de λ). Le tableau 5.5 résume les résultats obtenus dans les cas de classifications associées aux quatre valeurs suivantes de λ: 0,45, 0,46, 0,47 et 0,49. Nous avons choisi seulement ces quatre valeurs de λ pour, d'une part, simplifier notre illustration et, d'autre part, du fait que ces quatre valeurs fournissaient les meilleures précisions des estimations du réseau RBFN dans les expérimentations où $s_i$ était égale à 30 (Fig. 5.10). L'analyse des résultats du tableau 5.5 et de la figure 5.11 montre que quand $s_i$ est au voisinage de $R_0$, la précision des estimations fournies par le réseau RBFN sont satisfaisantes (parties hachurées sur le tableau 5.4). Cependant, quand $s_i$ est loin de $R_0$, le réseau génère des estimations moins précises. La marge de tolérance autour de $R_0$ dépend de la classification obtenue. En effet, si les *clusters* sont très compacts et dispersés, la marge de tolérance est assez grande.

Pour conclure cette validation, l'utilisation d'un réseau RBFN équivalent à *Fuzzy Analogy* nous permettra de retrouver automatiquement tous les paramètres relatifs aux valeurs linguistiques utilisées pour évaluer les attributs décrivant les projets logiciels. Ainsi, nous pourrons utiliser les valeurs de ces paramètres dans le processus d'estimation du *Fuzzy Analogy*. Cependant, l'apprentissage au niveau du quantificateur linguistique, utilisé pour évaluer la similarité globale entre deux projets logiciels, se fera selon la procédure de la figure 5.2.

**Tableau 5.5**
MMRE et Pred (25) des estimations d'un réseau RBFN en fonction de $s_i$

| | λ=0,45 $R_0$=44,01 | | λ=0,46 $R_0$=44,99 | | λ=0,47 $R_0$=45,97 | | λ=0,49 $R_0$=47,93 | |
|---|---|---|---|---|---|---|---|---|
| $s_i$ | MMRE | Pred(25) | MMRE | Pred(25) | MMRE | Pred(25) | MMRE | Pred(25) |
| **1** | 9,52 | 90,48 | 7,94 | 92,06 | 12,7 | 87,3 | 19,05 | 80,95 |
| **5** | 9,52 | 90,48 | 7,94 | 92,06 | 12,7 | 87,3 | 19,05 | 80,95 |
| **10** | 8,87 | 90,48 | 7,27 | 92,06 | 11,9 | 87,3 | 18,23 | 80,95 |
| **15** | 4,5 | 95,24 | 3,14 | 96,83 | 21,05 | 92,06 | 26,27 | 85,71 |
| **20** | 3,45 | 95,24 | 2,78 | 95,24 | 22,59 | 92,06 | 30,2 | 85,71 |
| **25** | 2,62 | 96,83 | 1,87 | 95,24 | 21,49 | 93,65 | 29,2 | 87,3 |
| **30** | 2,64 | 96,83 | 1,87 | 95,24 | 21,02 | 93,65 | 28,87 | 85,71 |
| **35** | 2,64 | 95,24 | 1,49 | 96,83 | 20,62 | 92,06 | 28,54 | 84,13 |
| **40** | 3,61 | 93,65 | 3,21 | 96,83 | 21,35 | 92,06 | 27,82 | 87,3 |
| **45** | 5,35 | 95,24 | 5,14 | 95,24 | 22,6 | 92,06 | 27,48 | 87,3 |
| **50** | 7,26 | 92,06 | 7,05 | 95,24 | 23,75 | 92,06 | 27,16 | 85,71 |
| **55** | 9,02 | 87,3 | 8,8 | 88,89 | 24,77 | 84,13 | 27,18 | 88,89 |
| **60** | 10,46 | 84,13 | 10,45 | 87,3 | 25,57 | 80,95 | 28 | 77,78 |
| **65** | 11,56 | 82,54 | 11,77 | 85,71 | 26,16 | 80,95 | 28,48 | 77,78 |
| **70** | 12,32 | 82,54 | 12,67 | 84,13 | 26,62 | 80,95 | 28,8 | 77,78 |
| **75** | 12,9 | 82,54 | 13,2 | 84,13 | 27,04 | 80,95 | 29,1 | 79,37 |

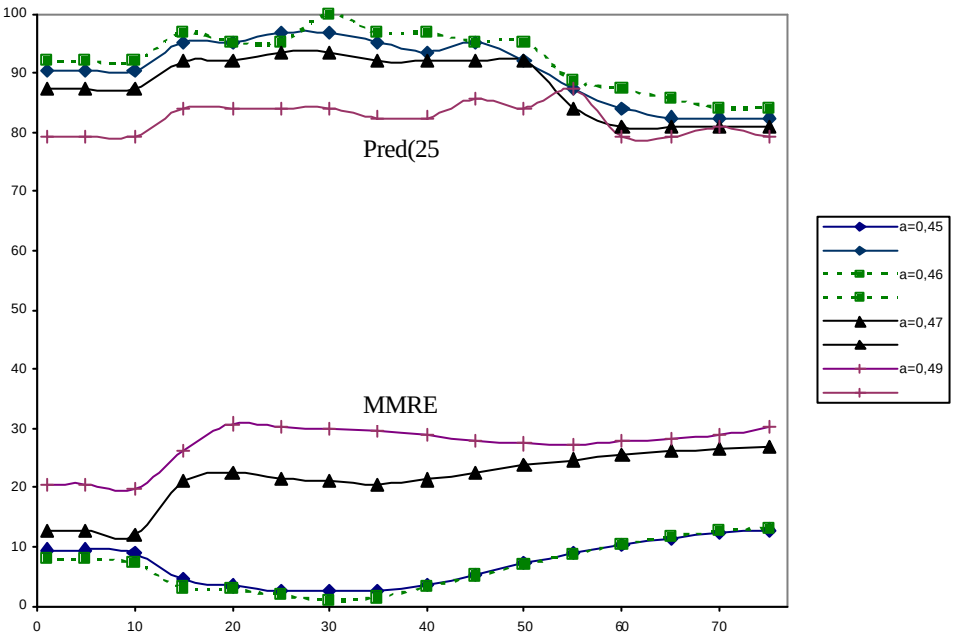



**Figure 5.11** MMRE et Pred(25) du réseau RBFN en fonction de σ pour les quatre valeurs de λ: 0,45, 0,46, 0,47 et 0,49.

## 5.6 DISCUSSION ET CONCLUSION

Dans ce chapitre, nous avons présenté et discuté de l'intégration de l'apprentissage dans notre approche d'estimation des coûts par analogie: *Fuzzy Analogy*. La caractéristique d'apprentissage permet de faciliter la reconfiguration et la modification des différents paramètres de *Fuzzy Analogy* afin qu'elle puisse s'adapter aux nouveaux changements survenus dans son environnement. La nécessité de cette caractéristique d'apprentissage dans *Fuzzy Analogy* provient du fait que l'industrie du logiciel est continuellement évolutive.

Ainsi, dans un premier temps, nous avons intégré dans le prototype logiciel F_ANGEL plusieurs fonctionnalités qui permettent à l'estimateur la mise à jour des différents paramètres de *Fuzzy Analogy*. L'estimateur pourra donc à travers F_ANGEL:

- Supprimer ou ajouter des attributs dans la description des projets logiciels;
- modifier les informations d'un attribut particulier (ajout et suppression de valeurs linguistiques, modification des fonctions d'appartenance des valeurs linguistiques et modification du poids associé à l'attribut),
- modifier la valeur linguistique *étroitement similaire*;
- mettre à jour la base de projets logiciels historiques; et
- modifier le quantificateur linguistique utilisé pour l'évaluation de la similarité globale entre deux projets.

Vu que la mise à jour de ces différents paramètres requiert une expertise de la part de l'estimateur, nous avons étudié la possibilité de rendre automatiques certaines de ces fonctionnalités. Ainsi, nous avons suggéré une procédure, utilisant le concept de la programmation génétique, pour la détermination des quantificateurs linguistiques appropriés à l'environnement étudié. Pour la mise à jour des différents paramètres relatifs aux attributs décrivant les projets logiciels, nous avons proposé l'utilisation des réseaux de neurones. Ainsi, nous avons étudié l'équivalence entre notre approche *Fuzzy Analogy* et différents types de réseaux de neurones. Tout d'abord, nous avons considéré le cas d'un Perceptron à trois

couches, avec l'algorithme de Rétro-propagation, auquel nous avons associé un système équivalent à base de règles floues. L'idée était d'établir indirectement, à travers ce système de règles floues, une équivalence entre le Perceptron à trois couches et notre approche *Fuzzy Analogy*. Cependant, nous n'avons pas pu fournir une interprétation compréhensible des règles floues obtenues. Par conséquent, nous avons abandonné cette idée et nous avons entrepris une étude pour établir une équivalence directe entre *Fuzzy Analogy* et les réseaux de neurones RBFN.

Nous avons ainsi élaboré un cadre théorique définissant les choix des différents paramètres d'un réseau RBFN (le nombre de neurones de la couche cachée, les fonctions d'activation et les règles d'apprentissage) et les choix de ceux de *Fuzzy Analgy* afin d'avoir une équivalence entre celui-ci et *Fuzzy Analogy*. Ces choix consistent en (Fig. 5.8):

- L'adoption d'un algorithme de classification pour déterminer le nombre de neurones de la couche cachée du réseau RBFN ainsi que les centres associés à ces neurones;
- l'utilisation de la fonction $f(x)=x^a$ comme fonction d'activation de tous les neurones de la couche cachée du réseau RBFN;
- l'incorporation de la fonction gaussienne pour la mesure des similarités individuelles entre les projets dans *Fuzzy Analogy* (Équation 5.7b);
- l'utilisation de la règle Delta pour la mise à jour des poids de la couche cachée à la couche de sortie du réseau RBFN;
- L'utilisation de la fonction Identité comme fonction d'activation du neurone de sortie du réseau RBFN.

Ensuite, nous avons validé le réseau RBFN, équivalent à *Fuzzy Analogy*, en utilisant les projets du COCOMO'81. Nous avons utilisé l'algorithme APC-III pour la classification des projets dans des clusters. Les classifications obtenues dépendent du paramètre $\lambda$ qui détermine le rayon $R_0$ utilisé par l'algorithme APC-III. Nous avons choisi celles qui permettent au réseau RBFN de fournir les meilleures estimations de coûts. Pour la détermination des valeurs $s_i$ qui serviront aux définitions des fonctions d'appartenance des

valeurs linguistiques utilisées dans *Fuzzy Analogy*, nous avons proposé de considérer une seule valeur pour toutes les $s_i$ et que cette valeur soit au voisinage de $R_0$.

# CONCLUSION

## 1. BILAN DU TRAVAIL

Dans ce travail de recherche, nous avons développé un modèle intelligent d'estimation des coûts de développement de logiciels *Fuzzy Analogy*. Ce modèle est basé sur un raisonnement par analogie auquel nous avons intégré la logique floue et les réseaux de neurones pour incorporer dans le modèle les trois caractéristiques principales de l'intelligence: 1) la tolérance des imprécisions, 2) la gestion des incertitudes, et 3) l'apprentissage. Le processus d'estimation de *Fuzzy Analogy* est composé de trois étapes:

- Identification des projets logiciels,
- évaluation de la similarité entre les projets logiciels, et
- utilisation des valeurs réelles des coûts des projets similaires pour déduire une estimation au coût du nouveau projet: étape d'Adaptation.

### 1.1 *Fuzzy Analogy* et la gestion des imprécisions

L'utilisation de la logique floue tout au long du processus d'estimation de *Fuzzy Analogy* nous a permis de traiter convenablement les imprécisions. Ainsi, dans l'étape d'identification des projets, nous avons représenté les valeurs linguistiques décrivant les projets par des ensembles flous plutôt que par des ensembles classiques. Cette représentation floue a été validée, dans un premier temps, en utilisant le modèle COCOMO'81 Intermédiaire. Nous avons défini les ensembles flous associés aux valeurs linguistiques (*très bas*, *bas*, *élevé*, *moyen*, *très élevé*) de 12, parmi 15, facteurs du COCOMO'81 Intermédiaire. Les résultats obtenus de cette validation ont montré que le *fuzzy* COCOMO'81 Intermédiaire est moins sensible, par rapport au COCOMO'81 Intermédiaire initial, à des petites variations dans ses entrées (Idri, Abran et Kjiri, 2000a; Idri et Abran, 2000b).

Dans l'étape d'évaluation de la similarité entre les projets logiciels, nous avons développé de nouvelles mesures de similarité pour traiter le cas des projets logiciels décrits par des valeurs linguistiques. Ces valeurs linguistiques sont représentées par des ensembles flous. Nos mesures évaluent la similarité globale entre deux projets $P_1$ et $P_2$, $d(P_1, P_2)$, en combinant les similarités individuelles $d_{v_j}(P_1, P_2)$. $d_{v_j}(P_1, P_2)$ utilisent des techniques d'agrégation floue (*max-min, sum-product, min-Kleene-Dienes*). $d(P_1,P_2)$ utilise des quantificateurs linguistiques monotones croissants (RIM) pour combiner les différentes similarités individuelles. Ainsi, nos mesures de similarité globale sont facilement adaptables aux besoins de l'environnement concerné du fait que leur adaptation ne se fait que par la modification de la définition du quantificateur linguistique utilisé. Nous avons validé ces mesures de similarité (individuelles et globales) en utilisant une approche axiomatique; les résultats de cette validation nous ont permis de choisir les mesures qui seront utilisées par *Fuzzy Analogy* pour la prédiction du coût d'un nouveau projet logiciel (Idri et Abran, 2000c, 2001a, 2001b).

Dans l'étape d'Adaptation du *Fuzzy Analogy*, nous avons redéfini les critères de sélection des projets *étroitement similaires* au nouveau projet ainsi que la formule utilisée pour déduire une estimation à son coût. Ainsi, nous avons utilisé une représentation floue de la valeur linguistique *étroitement similaire* afin d'éviter les critiques de la représentation classique utilisée auparavant en estimation des coûts par analogie. Par cela, nous avons donc intégré la gestion des imprécisions tout au long du processus d'estimation de *Fuzzy Analogy*. Pour valider notre modèle *Fuzzy Analogy*, nous avons comparé sa performance avec celles de trois autres modèles d'estimation: COCOMO'81 Intermédiaire, *fuzzy* COCOMO'81 Intermédiaire et l'analogie classique. Cette comparaison des performances considère deux critères: 1) la tolérance des imprécisions, et 2) la précision des estimations. Les résultats obtenus avantagent notre modèle et montrent l'utilité de la logique floue pour la gestion des imprécisions tout au long du processus d'estimation (Idri, Abran, Robert et Khoshgoftaar, 2001c, 2002b).

### 1.2 *Fuzzy Analogy* et la gestion des incertitudes

*Fuzzy Analogy* permet aussi la gestion des incertitudes au niveau des estimations de coûts. Pour ce faire, nous avons examiné deux sources d'incertitudes dans *Fuzzy Analogy*: 1) incertitudes relatives aux erreurs de mesurage des attributs décrivant les projets logiciel, et 2) incertitudes relatives aux imprécisions des estimations de *Fuzzy Analogy*. Nous avons montré que les incertitudes, relatives aux erreurs de mesurage des attributs, peuvent être masquées au cours des trois étapes de *Fuzzy Analogy*. Ainsi, nous nous sommes plus intéressés aux incertitudes relatives aux imprécisions des estimations de *Fuzzy Analogy*. Ces imprécisions proviennent de la nature de l'affirmation de base de *Fuzzy Analogy*: *Similar software projects have similar costs*. Nous avons donc mené une expérimentation sur la base de projets COCOMO'81 pour identifier le type de cette affirmation: déterministe ou non-déterministe. Nous avons conclu de cette expérimentation que l'utilisation de la version déterministe mène souvent à des situations où *Fuzzy Analogy* ne génère aucune valeur estimée au coût d'un nouveau projet. Nous avons donc opté pour utiliser la version non-déterministe afin de gérer les incertitudes au niveau des estimations fournies par notre modèle. Nous avons modélisé cette version non-déterministe: *Similar software projects have possibly similar costs*, par la théorie de possibilité. Ainsi, *Fuzzy Analogy* génère un ensemble de valeurs estimées, avec une fonction de distribution des possibilités, au coût de tout nouveau projet. Cette fonction de distribution peut être aussi utilisée pour gérer les risques associés aux incertitudes des estimations (Idri, Khoshgoftaar et Abran, 2002d).

### 1.3 *Fuzzy Analogy* et l'apprentissage

Afin de permettre à notre modèle de s'adapter facilement à son environnement, nous y avons intégré des mécanismes d'apprentissage. Ainsi, dans un premier temps, nous avons intégré dans le prototype F_ANGEL un ensemble de fonctionnalités qui permettent la mise à jour des différents paramètres de *Fuzzy Analogy*. Ces paramètres dépendent, en général, de l'état de l'environnement concerné. L'estimateur pourra donc modifier les valeurs de tous les paramètres de *Fuzzy Analogy* tels que les définitions des valeurs linguistiques, les pondérations des attributs, et la définition du quantificateur linguistique utilisée afin de satisfaire les nouveaux besoins de son environnement.

Dans une deuxième étape, nous avons étudié la possibilité d'automatiser la mise à jour de certains paramètres de *Fuzzy Analogy*, spécifiquement ceux relatifs aux valeurs linguistiques décrivant les projets logiciels. Ainsi, nous avons étudié l'établissement d'une équivalence entre *Fuzzy Analogy* et deux types particuliers de réseaux de neurones: Perceptron et RBFN. Nous avons commencé par étudier l'équivalence entre *Fuzzy Analogy* et un Perceptron à trois couches et cela via un système de règles floues. Malheureusement, les règles floues obtenues de la validation du Perceptron à trois couches sur les projets du COCOMO'81, n'étaient pas facilement interprétables (Idri, khoshgoftaar et Abran, 2002a; Idri, Abran et Mbarki, 2002c). Nous avons donc abandonné cette piste et nous avons étudié l'équivalence entre *Fuzzy Analogy* et un réseau RBFN. Nous avons établi un cadre théorique qui détermine les conditions suffisantes pour avoir une équivalence entre le processus de *Fuzzy Analogy* et le réseau RBFN. Nous avons validé le réseau RBFN équivalant à *Fuzzy Analogy*, sur les projets du COCOMO'81. Cette validation nous a permis de déterminer les valeurs adéquates de tous les paramètres de *Fuzzy Analogy* en relation avec les valeurs linguistiques.

## 2. CONTRIBUTIONS DE CE TRAVAIL DANS D'AUTRES DOMAINES DE RECHERCHE

Ce travail revêt un caractère multidisciplinaire. Ainsi, ses contributions vont au-delà du domaine de l'estimation des coûts de développement de logiciels. En particulier, nous citons:

- *Raisonnement à base de cas (CBR)*: La technique CBR a été appliquée dans plusieurs domaines de recherche, notamment en estimation des coûts. Par conséquent, chaque domaine a contribué à l'évolution et à la popularité de la technique CBR, par le développement de nouveaux algorithmes qui peuvent être utilisés dans les différentes étapes du processus standard CBR. Notre contribution consiste à développer de nouvelles mesures de similarité. Ces mesures ne sont pas spécifiques au cas des projets logiciels; elles peuvent être utilisées dans d'autres domaines, notamment pour la prédiction d'autres attributs logiciel tels que la qualité et la fiabilité.

- *Intelligence artificielle et logique floue***:** De plus en plus la contribution de la logique floue au développement des systèmes intelligents est évidente. En effet, elle offre plusieurs outils pour traiter convenablement les informations imprécises et incertaines. Une telle caractéristique est nécessaire pour tout système intelligent qui opère dans des domaines où la connaissance est imprécise et incertaine. En étudiant le cas du domaine d'estimation du coût, notre travail illustre le rôle important de la logique floue au sein de l'intelligence artificielle. De plus, dans ce travail, nous avons utilisé une panoplie de techniques, relevant des deux approches classiques en sciences cognitives: Symbolisme et Connexionnisme, pour développer notre modèle intelligent. Ceci est conforme et soutient les nouvelles tendances en intelligence artificielle encourageant une approche hybride plutôt qu'une approche purement symboliste ou connexionniste, pour la résolution de problèmes complexes (Tirri, 1991; Towell, 1992; Medsker, 1994; Jang, Sun et Mizutani, 1997).

- *Métrologie de logiciels***:** La métrologie de logiciels est l'application directe de la théorie de mesures développée par Krantz et al. (1971). Cette théorie considère souvent que les résultats des mesures sont des valeurs numériques. En génie logiciel, l'utilisation des valeurs numériques, pour l'évaluation des attributs d'un logiciel, ne nous semble pas toujours appropriée. En effet, les évaluations de la plupart des attributs d'un logiciel sont souvent subjectives. Par conséquent, les résultats de ces évaluations (mesures) sont des valeurs linguistiques plutôt que des valeurs numériques. Afin de représenter convenablement les imprécisions qui engendrent ces valeurs linguistiques, elles doivent être représentées par des ensembles flous. Notre travail explore cette issue en métrologie de logiciels et montre les avantages de l'utilisation des valeurs linguistiques comme résultats du processus de mesurage des attributs logiciels.

- *Computation intelligente en génie logiciel*: La computation intelligente (*Computational Intelligence*) vise principalement la simulation de l'intelligence des humains par des mécanismes de computation (Jordan et Russell, 1999). Son utilisation en génie logiciel a pour objectif l'amélioration des paradigmes classiques de développement, de maintenance et/ou de contrôle de logiciels, afin de répondre

aux nouveaux besoins émanant de domaines qui utilisent des logiciels intelligents tels que la robotique et le contrôle des processus industriels. Récemment, plusieurs travaux de recherche ont été menés pour montrer l'utilité de la computation intelligente en génie logiciel (Ebert, 1993; Gray et MacDonell, 1997; Khoshgoftaar et Allen, 1997; Khoshgoftaar et al., 1997; Pedrycz et Peters, 1997; Pedrycz et Sosnowski, 2001). Notre travail confirme donc cette tendance vu les résultats encourageants obtenus dans cette recherche.

## 3. PERSPECTIVES

Le cadre de recherche que nous avons choisi pour cette thèse est relativement large. Nous n'avons aucunement la prétention de l'avoir totalement abordé. Ainsi, de nombreux travaux voire programmes de recherche peuvent être envisagés dans cette perspective. Tout d'abord, nous identifions deux perfectionnements mineurs à apporter à la procédure d'apprentissage de *Fuzzy Analogy*:

- Analyse de la relation existante entre la valeur linguistique *étroitement similaire* et la règle d'apprentissage Delta utilisée dans le réseau RBFN, et
- interprétation des ensembles flous obtenus à partir du réseau RBFN.

Ensuite, nous comptons apporter certaines améliorations à la stratégie d'apprentissage adoptée par *Fuzzy Analogy*. Nous suggérons d'expérimenter d'autres algorithmes de classification dans le réseau RBFN équivalant à notre modèle *Fuzzy Analogy*. En effet, l'algorithme APC-III que nous avons utilisé présente deux difficultés:

1- Il n'intègre pas la procédure d'apprentissage du réseau RBFN. En effet, cette procédure concerne seulement les poids de la couche cachée à la couche de sortie. Par conséquent, la procédure d'apprentissage de APC-III doit être validée avant et indépendamment de celle du réseau RBFN.

2- Il utilise un paramètre $\lambda$ qui doit être déterminé empiriquement. Ce paramètre $\lambda$ est décisif pour la détermination des clusters à partir de la base de projets logiciels.

Afin de remédier à ces deux difficultés, nous proposons l'utilisation de l'algorithme de Kohonen pour la classification des projets logiciels. En effet, l'algorithme de Kohonen réalise un apprentissage compétitif d'une manière computationnelle. Sa règle d'apprentissage permet de retrouver les meilleurs poids $w_{ij}$ de la couche d'entrée à la couche cachée:

$$\Delta w_{ij} = \eta S(i, i^*)(e_j - w_{ij}) \qquad \forall i, j$$

où $\eta$ est le taux d'apprentissage, $S$ est une mesure de similarité entre le neurone i et le neurone gagnant $i^*$, et $e_j$ représente le projet à classifier. La convergence de l'algorithme génère les poids $w_{ij}$ représentant les composantes des centres de la classification obtenue. La difficulté de l'utilisation de l'algorithme de Kohonen dans un réseau RBFN réside dans le contrôle de la convergence de toute la procédure d'apprentissage du RBFN. Cette procédure est composée de deux sous-algorithmes d'apprentissage: 1) l'algorithme de Kohonen pour retrouver les poids $w_{ij}$ de la couche d'entrée à la couche cachée et 2) la règle Delta pour retrouver les poids $b_j$ de la couche cachée à la couche de sortie. Des travaux de recherche et d'expérimentation sont actuellement en cours pour trouver une solution à cette difficulté.

Nous suggérons aussi une autre amélioration de la procédure d'apprentissage utilisée dans *Fuzzy Analogy*, spécifiquement au niveau du quantificateur linguistique utilisé pour l'évaluation de la similarité globale entre les projets logiciels. En effet, nous avons proposé d'utiliser la programmation génétique pour déterminer le quantificateur approprié à l'environnement étudié. Nous n'avons pas validé cette procédure pour le cas de la base COCOMO'81 du fait que cette base de projets ne présentait pas de difficultés pour la détermination de son quantificateur adéquat. Par conséquent, nous envisageons tout d'abord de valider cette procédure dans les cas d'autres bases de projets logiciels; ensuite, nous proposons d'examiner la possibilité d'intégrer cette procédure dans le réseau RBFN afin de retrouver facilement les fonctions d'activation des neurones de la couche cachée.

Enfin, pour une large acceptation de notre modèle d'estimation, d'autres validations empiriques sur des bases contenant un nombre élevé de projets logiciels sont nécessaires. Spécifiquement, nous comptons utiliser la base de projets ISBSG (*International Software Benchmarking Standards Group*) qui contient déjà des données sur plus que 2 000 projets logiciels.

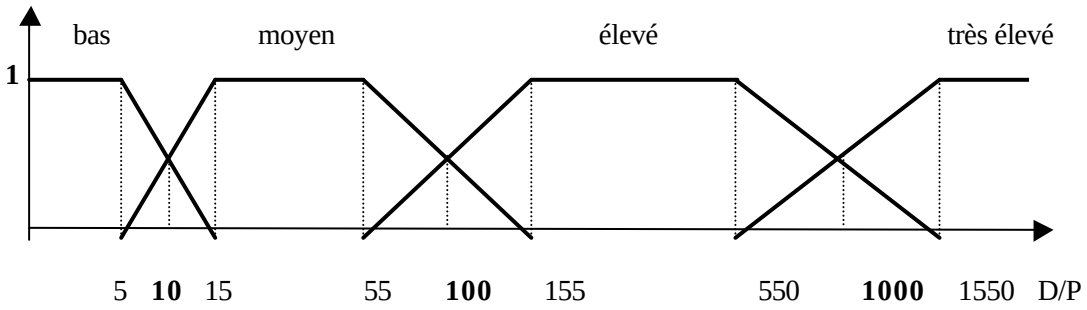
## APPENDICE A

## ENSEMBLES FLOUS CORRESPONDANT AUX VALEURS LINGUISTIQUES DU MODÈLE COCOMO'81

Cet appendice présente les ensembles flous associés aux valeurs linguistiques des 12 facteurs du modèle COCOMO'81. Pour chaque facteur, nous présentons tout d'abord une brève description; ensuite, nous présentons les ensembles flous associés à ses valeurs linguistiques. Les ensembles flous de tous les facteurs sont représentés par des fonctions d'appartenance trapézoïdales et forment, pour chaque facteur, une répartition floue.

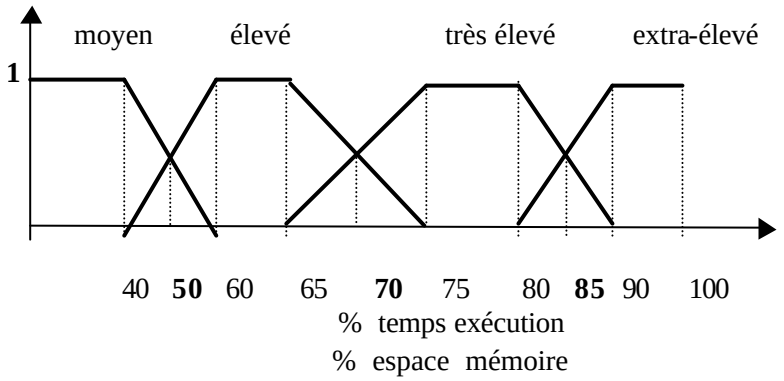
### DATA (Data Base Size)

Ce facteur exprime l'influence de la taille de la base de données sur le coût de développement du logiciel. Ainsi, plus la taille de la base de données est grande plus elle influence positivement le coût du logiciel. DATA est mesuré par le ratio D/P où D est la taille de la base de données et P est la taille du code source du logiciel mesurée en ISL. Le facteur DATA est évalué par quatre valeurs linguistiques: *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



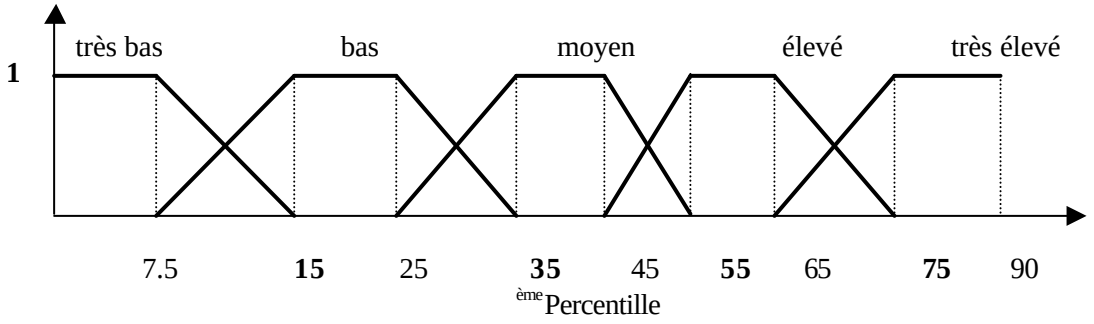
**TIME et STOR (*Execution Time Constraint, Main Storage Constraint*)**

Le facteur TIME (STOR) exprime l'effet des ressources temporelles (spatiales) exigées par le logiciel sur le coût de développement. Ainsi, plus ces ressources sont considérables, plus elles influencent positivement le coût de développement du logiciel. TIME (STOR) est mesuré en termes de pourcentage d'utilisation des ressources temporelles (spatiales) disponibles. Le facteur TIME (STOR) est évalué par quatre valeurs linguistiques: *moyen*, *élevé*, *très élevé* et *extra-élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



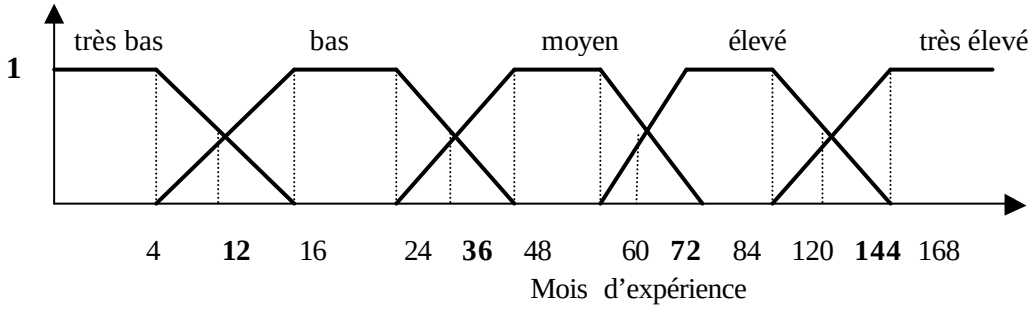

**ACAP et PCAP (*Analyst Capability, Programmer Capability*)**

Le facteur ACAP (PCAP) exprime l'effet de la compétence des analystes (programmeurs) sur le coût de développement du logiciel. Ainsi, plus cette compétence est faible, plus le coût de développement augmente. ACAP (PCAP) est mesuré par l'ordre de classement des analystes (programmeurs). Le facteur ACAP (PCAP) est évalué par cinq valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



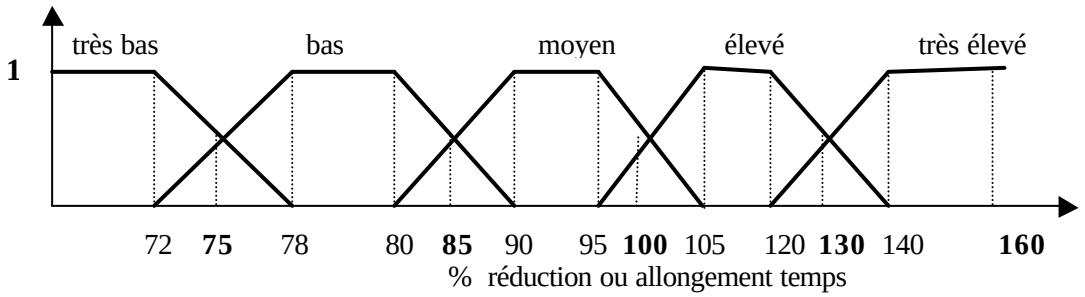
**AEXP (*Application Experience*)**

Le facteur AEXP exprime l'effet de l'expérience du personnel impliqué dans développement, sur le coût du logiciel. Ainsi, plus cette expérience est élevée, plus le coût de développement est faible. Le facteur AEXP est mesuré par le nombre d'années d'expérience. AEXP est évalué par cinq valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



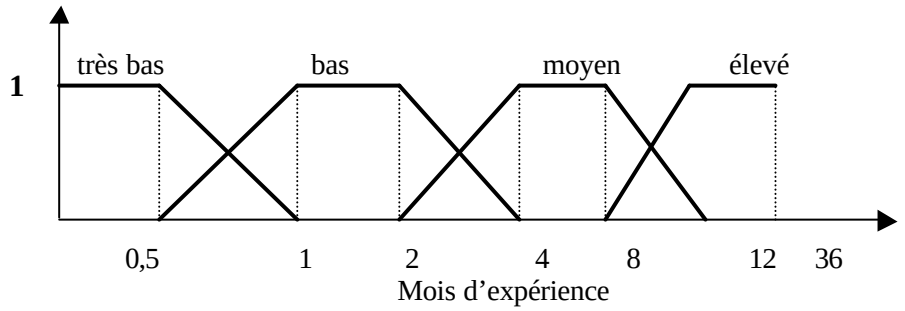

**SCED (*Required Development Schedule*)**

Le facteur SCED exprime l'effet des contraintes de réduction ou d'allongement du temps de développement initialement prévu, sur le coût de développement du logiciel. Il génère une augmentation du coût dans les deux cas: réduction ou allongement du temps. SCED est mesuré par le pourcentage de réduction ou d'allongement du temps de développement initial. Le facteur SCED est évalué par cinq valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



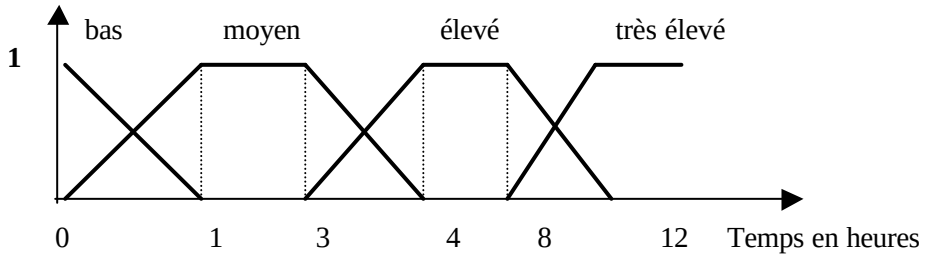
**VEXP & LEXP (*Virtual Machine Experience, Programming Language Experience*)**

Le facteur VEXP (LEXP) exprime l'effet de l'expérience du personnel dans l'utilisation de la machine virtuelle (langage de programmation) sur le coût de développement du logiciel. Ainsi, plus cette expérience est élevée, plus le coût de développement diminue. VEXP (LEXP) est mesuré par le nombre d'années d'expérience. Le facteur VEXP (LEXP) est évalué par cinq valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.




**TURN (*Computer Turnaround Time*)**

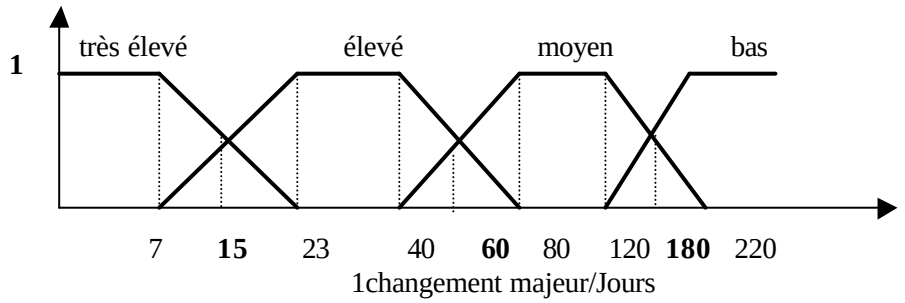
Le facteur TURN exprime l'effet du temps de réponse du système sous lequel on développe le logiciel, en rapport avec le coût de développement du logiciel. Ainsi, plus le temps de réponse est élevé, plus le coût de développement augmente. TURN est mesuré par le nombre d'heures d'attente pour avoir la réponse du système. Le facteur TURN est évalué par quatre valeurs linguistiques: *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.



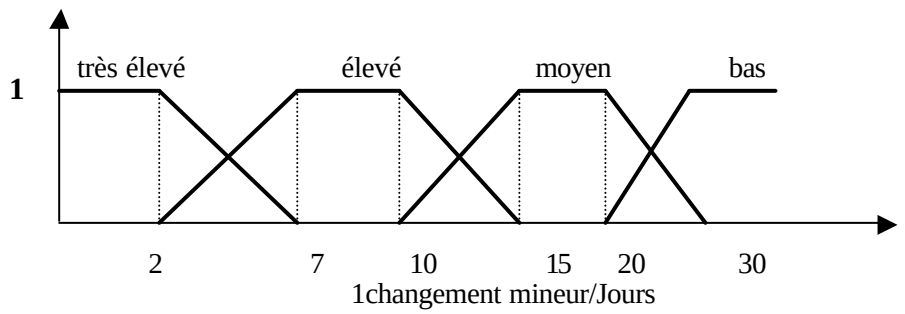
**VIRT (*Virtual Machine Volatility*)**

Le facteur VIRT exprime l'effet de la volatilité de la machine virtuelle sur le coût de développement du logiciel. Le coût de développement augmente si la machine virtuelle est très volatile. VIRT est mesuré par la fréquence des changements apportés à la machine virtuelle durant le développement du logiciel. Boehm distingue deux types de changements: *mineur* et *majeur*. Ainsi, nous avons associé à chaque type de changements un facteur: VIRT-mineur et VIRT-majeur. Le facteur VIRT-mineur (VIRT-majeur) est évalué par quatre valeurs linguistiques: *très bas*, *bas*, *moyen*, *élevé* et *très élevé*. Leurs représentations par des ensembles flous sont illustrées ci-dessous.

VIRT-majeur




VIRT-mineur



# APPENDICE B

## QUELQUES ÉCRANS DU PROTOTYPE LOGICIEL F_ANGEL

Cet appendice présente quelques écrans du prototype logiciel F_ANGEL. F_ANGEL a été développé avec MS Visual Basic 6.0, pour implémenter les différents traitements du processus d'estimation de *Fuzzy Analogy*, et MS Access pour le stockage de données sur les projets logiciels. En plus de la fonctionnalité principale de F_ANGEL, à savoir l'estimation du coût d'un logiciel, le prototype fournit plusieurs autres fonctionnalités relatives à la mise à jour de la plupart des paramètres de notre modèle:

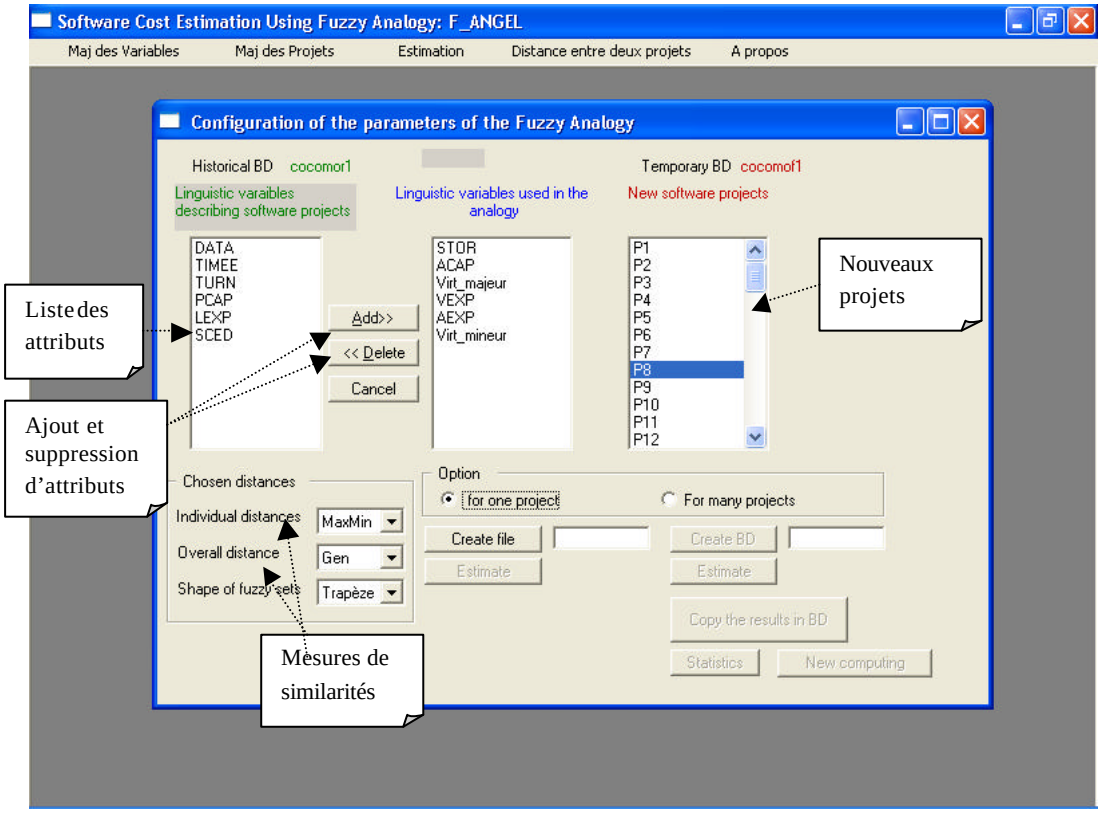
- Mise à jour des paramètres relatifs aux attributs décrivant les projets logiciels,
- mise à jour de la base de projets logiciels, et
- analyse des similarités entre les projets logiciels.

### ÉCRAN D'ESTIMATION DES COÛTS

Cet écran permet à l'estimateur de choisir les différentes valeurs des paramètres du processus d'estimation de *Fuzzy Analogy.* Ainsi, l'estimateur pourra choisir:

- Les attributs qui serviront à la description des projets logiciels. Cet ensemble d'attributs est variable d'une estimation à une autre.
- La mesure de similarité individuelle (*max-min* ou *sum-product*).
- La mesure de similarité globale (la fonction $f(x)=x^{\alpha}$).

Le nouveau projet, pour lequel on veut estimer le coût, doit être placé auparavant dans la base de projets temporaires (non-achevés). L'estimateur pourra exécuter F_ANGEL sur un ou plusieurs nouveaux projets en les sélectionnant sur la liste correspondante à la base de projets non-achevés. Les résultats d'estimation du coût d'un nouveau projet sont enregistrés dans un fichier. Dans le cas de plusieurs nouveaux projets, les résultats sont enregistrés dans une base de données.



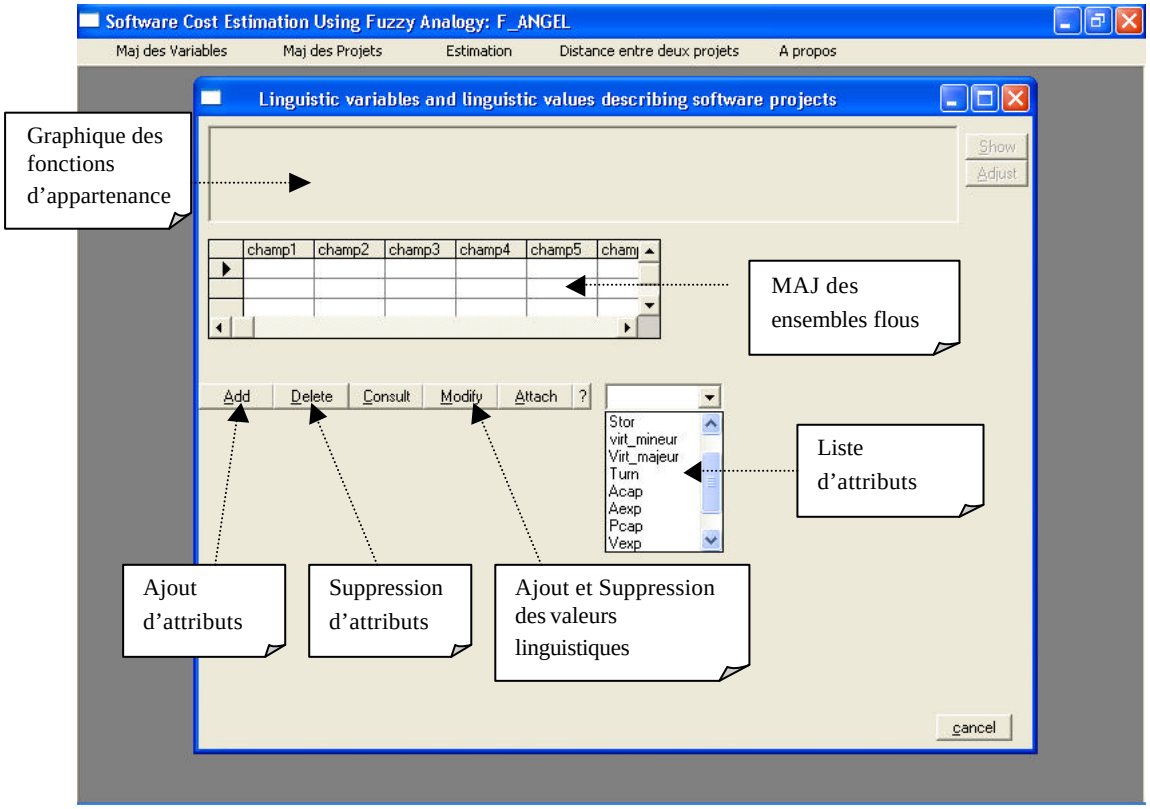

**Écran du processus d'estimation**

## ÉCRAN DE MISE À JOUR DES ATTRIBUTS

Cet écran permet à l'estimateur de modifier les différents paramètres relatifs aux attributs décrivant les projets logiciels. L'objectif de la modification de ces paramètres est de permettre à notre modèle de prendre en considération les nouveaux changements survenus dans le domaine étudié. Des exemples de ces mises à jour sont:

- Ajout et/ou suppression d'attribut,
- modification des définitions des différents ensembles flous représentant les valeurs linguistiques des attributs,
- ajout et/ou suppression des valeurs linguistiques, et
- modification des pondérations associées aux attributs.

Toutes les modifications apportées aux différents paramètres relatifs aux attributs seront prises en considération dans la prochaine formulation d'une estimation au coût d'un nouveau projet.

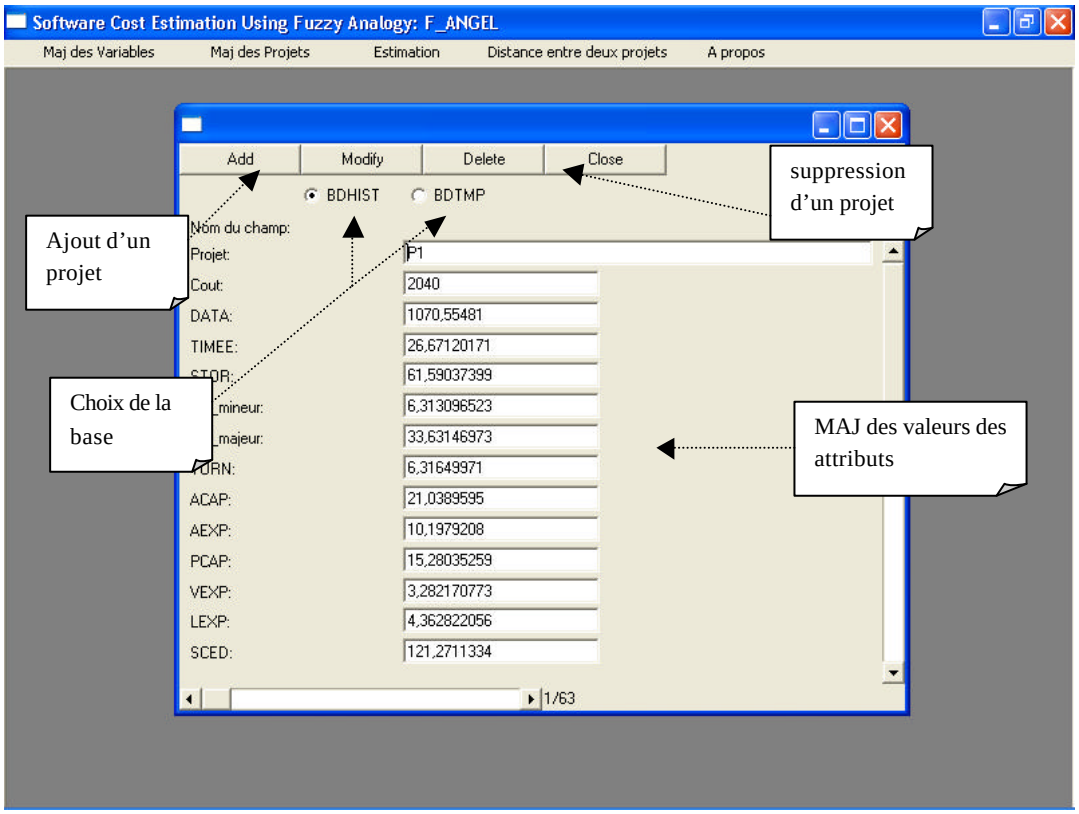



**Écran pour la modification des paramètres relatifs aux attributs**

## ÉCRAN DE MISE À JOUR DES PROJETS LOGICIELS

Cet écran permet de modifier les deux bases de projets logiciels: projets logiciels achevés (projets historiques) et projets logiciels non-achevés (projets temporaires). Ainsi, l'estimateur pourra:

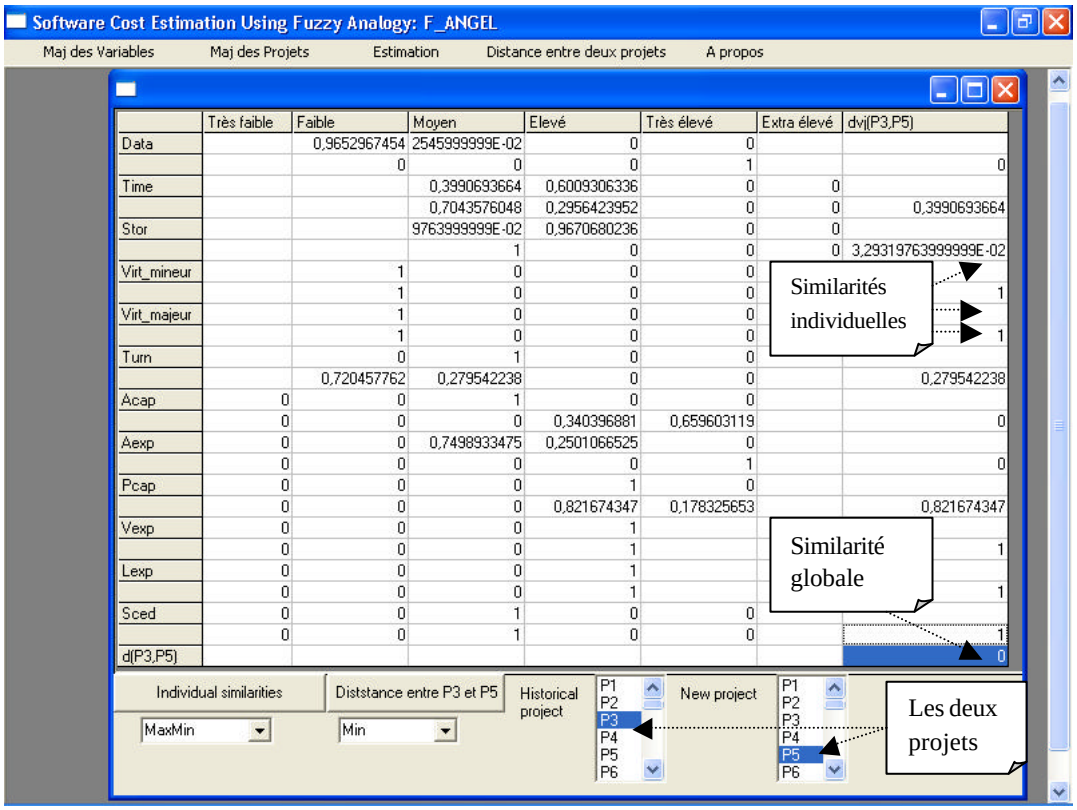
- Ajouter ou supprimer un projet de la base historique et/ou temporaire,
- modifier les valeurs des attributs d'un projet de l'une des deux bases, et
- Ajouter un projet de la base temporaire, quand il est achevé, à la base de projets historiques.




**Écran pour la mise à jour des deux bases de projets historiques et temporaires**

## ÉCRAN D'ANALYSE DES SIMILARITÉS ENTRE LES PROJETS LOGICIELS

Cet écran permet l'analyse des valeurs de similarité entre deux projets logiciels. Cette analyse est importante dans certaines situations où l'estimateur aura besoin de plus de détails sur le processus d'estimation de *Fuzzy Analogy*. Ainsi, l'estimateur pourra analyser de près les similarités entre un nouveau projet et certains projets historiques.




**Écran d'analyse des similarités entre deux projets logiciels**

**Appendice C**

**COPIES DE TROIS ARTICLES DE RÉFÉRENCES**

## BIBLIOGRAPHIE


Aamodt, Agnar, et Enric Plaza. 1994. «Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches». *AI Communications*, IOS Press, vol. 7, no 1, p. 39-59.

Abran, Alain, et P.N. Robillard. 1996. «Functions Points Analysis: An Empirical Study of its Measurement Processes». *IEEE Transactions on Software Engineering*, vol. 22, no. 12, p. 895-909.

Aha, W. D. 1991. «Case-Based Learning Algorithms». *DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann, LoaAtlos, CA.

Amrane, Bakhta, et Yhaya Slimani. 1993. «ALCOMO Model: A Statistical Reformulation of the COCOMO'81 Model». *Information science and Technology*, Avril.

Angelis, L., et I. Stamelos. 2000. «A Simulation Tool for Efficient Analogy Based Cost Estimation». *Empirical Software Engineering*, vol. 5, no. 1, p. 35-68.

Angelis, L., I. Stamelos, et M. Morisio. 2001. «Building a Software Cost Estimation Model Based on Categorical Data». *Proceedings of the 7th International Software Metrics Symposium*, 2-7 Avril, IEEE Computer Society, p. 4-15.

Albrecht, A.J., et J.E. Gaffney. 1983. «Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation». *IEEE Transactions on Soft Eng*, vol. SE-9, no. 6, Novembre, p. 639-647.

Bartsch-Spoerl, B. 1995. «Towards the Integration of Case-based, Schema-based, and Model-based Reasoning for Supporting Complex Design Tasks». *Proc. 1st International Conference on case-based reasoning*, p. 145-156

Benitez, J. M., J. L. Castro, et I. Requena. 1997. «Are Artificial Neural Networks Black Boxes». *IEEE Transactions on Neural Networks*, vol. 8, no. 5, Septembre, p. 1156-1164.

Bell, B., S. Keddar, et R. Bareiss. 1994. «Interactive Model-driven Case Adaptation for instructional Software Design» *Proc. 16th Annal Conference of the Cognitive Science* Society, Lawrence Erlbaum, p. 33-38.

Bisio, R., et F. Malabocchia. 1995. «Cost Estimation of Software Project through Case Base Reasoning». *First International conf. on Case-Based reasoning*, Sesimbra, Springer, p.11-22.

Boehm, Barry W. 1981. *Software Engineering Economics*, Prentice-Hall.

Boehm, Barry W., B. Clark, E. Horowitz, C. Westland, R. Madachy, R. Selby. 1995. «Cost Models for Future Software Life Cycle Processes: COCOMO II». *Annals of Software Engineering: Software Process and Product Measurement*, Amsterdam,.

Box, G. E. P., D. R. Cox. 1964. « An Analysis of transformations ». *Journal Royal Statistical Society*, vol. *26*, Serie B., p. 211-252.

Breiman L., J. Friedman, R. Olshen et C. Stone. 1984. *Classification and regression trees*. Belmont, CA: Waldsworth International.

Briand, Lionel, T. Langley, et I. Wieczorek. 2000. «Using the European Space Agency Data Set: A Replicated Assessment and Comparison of Common Software Cost Modeling». *Proceedings of the 22nd International Conference on Software Engineering*, Limerik, Irelande, p. 377-386.

Briand, Lionel, K. El Emam, K. Maxwell, D. Surmann, et I. Wieczorek. 1999. « An assessment and comparison of common software cost estimation models». *Proceedings of the 21nd International Conference on Software Engineering*, Los Angeles, p. 313-322.

Buckley J.-S., Y. Hayashi, et E. Czogala. 1993. «On the Equivalence of Neural Nets and Fuzzy Expert Systems». *Fuzzy Sets Systems*, no. 53, p. 129-134.

Burgess, Colin J., et Martin Lefley. 2001. «Can genetic programming improve software effort estimation? A comparative evaluation». *Information and Software Technology*, vol. 43, p. 863-873.

Chang, Carl K., M. J. Christensen et Zhang T. 2001. «Genetic Algorithms for project management ». *Annals of Software Engineering*, Kluwer Academic Publishers, vol. 11, p. 107-139.

Chaudet ,H. et L. Pellegrin. 1998. *Intelligence artificielle et psychologie cognitive*. Dunod.

Cheeseman Peter. 1985. «In defense of probability». *Proceedings of the 9th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers. Los Angels, p. 1002-1009.

Chulani, D.S. 1998. «Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and Its Quality Model Extension». Ph.D. Qualifying Exam Report, University of Southern California.

Conte S. D., H. D. Dunsmore et V. Y. Shen. 1986. *Software Engineering Metrics and Models*, Benjamin-Cummings. Menlo Park, CA

Davalo, Eric, et Patrick Naïm. 1990. *Des réseaux de neurones*, Eyrolles.

Davis, L. 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Debuse, J. C. W., et V. J. Rayward-Smith. 1997. «Feature Subset Selection within a Simulated Annealing Data Mining Algorithm». *Journal of Intelligent Information Systems*, vol. 9, p. 57-81.

DeMarco, Tom. 1982. *Controlling Software Projects*. Yourden Press. Ney York.

Desharnais Jean Marc. 1989. «Analyse statistique de la productivité des projets informatiques à partir de la technique des points de fonctions». Mémoire de Maîtrise, Université de Montréal.

Dolado, J. J. 2000. «On the Problem of the Software Cost Functions». *Information and Software Technology*, vol. 00, p. 1-12.

Draper N. R. and H. Smith. 1981. *Applied Regression Analysis*. John Wiley & Sons, Ney York.

Dubois, Dedier, et H. Prade. 1980. *Fuzzy Set Teory and Systems*, Ney York Academic.

Dubois, Dedier, et H. Prade. 1991. « Fuzzy sets in approximate reasoning, part 1: Inference with possibility distributions ». *Fuzzy sets and Systems*, vol. 40, p.143-202.

Dubois, Didier, F. Esteva, P. Garcia, L. Godo, R. L. deMantaras et H. Prade. 1999. «Case-based Reasoning: A Fuzzy Approach». *Workshop on Fuzzy Logic in Artificial Intelligence fonctions . Lecture Notes in Atificial Intelligence*, vol. 1566, Springer, Berlin, p. 79-90.

Ebert, C. 1993. «An Approach to Fuzzy Data Analysis for Software Quality Models». *Proc. 1st European Congress on Fuzzy and Intelligent Technologies*, Aachen, Septembre, p.1156-1161.

Elkan C. 1993. «The Paradoxical Success of Fuzzy Logic». Proceedings of the National *Artificial Intelligence Conference*. AAAI: MIT Press, p.698-703.

Fenton, Norman, et S.L. Pfleeger. 1997. *Software Metrics: A Rigorous and Practical Approach*. International Computer, Thomson Press.

Ganesan, K., T. M. Khoshgoftaar et E. Allen. 2000. «Case-Based Software Quality Prediction». *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 2, p.139-152.

Gallant, S. I. 1993. *Neural Network Learning and Expert Systems*, MIT Press, Cambridge.

Gentner, G. 1983. «Structure Mapping: A Theorotical Framework of Analogy». *Cognitive Science*, vol. 7, p. 155-170.

Gentner, D., et A. Markman. 1997. «Structure Mapping in Analogy and Similarity». *American Psychological Association*, vol.. 52, no. 1, p. 45-56.

Gentner, G., 1998. «Analogy». *In W. Bechtel & G. Graham (Eds), A companion to cognitive science*, Oxford: Blackwell, p. 107-113.

Gentner, D., K. J. Holyoak et B. Kokinov. 2001. *The Analogical Mind: Perspectives from Cognitive Science*, Bradford Books, Mars.

Gokhale S. S., M. R. Lyu. 1997. «Regression Tree Modeling for the Prediction of Software Quality». *Proceedings of the Third International Conference on Reliability and Quality in Desing*, International Society of Sciences and Applied Technologies, p. 31-36.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimisation and Learning*. Addison Wesley, Reading, MA.

Gulezian., G. 1991. «Reformulating and Calibrating COCOMO». *Journal Systems Software*, vol. 16, p.235-242.

Gray, A. R., et S. G. NacDonell. 1997. «Fuzzy Logic Techniques For Software Metric Models of Development». *Advances in Fuzzy Systems-Applications and Theory*: *Computational Intelligence in Software Engineering*, Editeurs: W. Pedrycz et J.F. Peters, vol. 16, p. 321-338.

Hayken, S. 1994. *Neural Networks: A Guide to Intelligent Systems*, Wesley.

Henri, S., D. Cafura. 1981. «Software structure metrics based on information flow ». *IEEE Transaction on Software Engineering*. Vol. 7, No. 5. p. 510-518.

Hertz, John, Anders Krogh et Richard G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Addison Wesley.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Holyoak, J. K., et P. Taghard, 1995.*Mental Leaps: Analogy in Creative Thought.* MIT Press.

Hornik, K., M. Stinchcombe et H. White. 1989. «Multilayer Feedforward Networks are Universal Approximators». *Neural Networks*, vol. 2. p. 359-366

Hughes, R.T. 1996. «An Evaluation of Machine Learning Techniques for Software Effort Estimation». Mater thesis, University of Brighton.

Hüllermeier, E. 2001. «Similarity-based inference as evidential reasoning». *International Journal of Approximate Reasoning*, vol. 26, p. 67-100.

Hwang Y.-S., et S.-Y. Bang.1997. «An Efficient Method to Construct a Radial Basis Function Networks Classifier». *Neural Networks*, Vol. 10, no. 08, p. 1495-1503.

Idri, Ali. 1997. «Etude Préalable pour la mise au point d'un modèle d'estimation du coût de développement de logiciels : le projet MOCOMO». Thèse Doctorat de troisième cycle, Université Mohamed V, Rabat, Juillet, p. 190.

Idri, Ali, Brahim Griech et Abdelhamid El-Iraki. 1997. «Towards an Adaptation of the COCOMO to the Software Measurement Theory». *6ème European Software Engineering Conference et 11ème ACM Symposium on Foundations of Software Engineering* , Springer, Zurich, Septembre, p. 525-526.

Idri, Ali, Alain Abran et Laila Kjiri. 2000a. «COCOMO Cost Model Using Fuzzy Logic». *Proceedings of the 7th International Conference on Fuzzy Theory and Technology*, Atlantic City, février, p. 219-223.

Idri, Ali, et Alain Abran, 2000b. «La mise en valeur d'une approche floue pour l'évaluation du coût de développement de logiciel», *CARI*, Madagacscar, Octobre, p.247-254.

Idri, Ali, et Alain Abran. 2000c. «Towards A Fuzzy Logic Based Measures for Software Project Similarity». *Proceedings of the 6th Maghrebian Conference on Computer Sciences*, novembre, Fes, Maroc, p. 9-18.

Idri, Ali, et Alain Abran. 2001a. «A Fuzzy Logic Based Measures For Software Project Similarity: Validation and Possible Improvements», *Proceedings of the 7th International Symposium on Software Metrics*, avril, Londres, IEEE Computer Society, p. 85-96.

Idri, Ali, et Alain Abran. 2001b. «Evaluating Software Projects Similarity by Using Linguistic Quantifier Guided Aggregations». *Proceedings of the 9th IFSA World Congress and 20th NAFIPS International Conference*, juillet, Vancouver, p. 416-421.

Idri, Ali, Alain Abran, et Taghi Khoshgoftaar. 2001c. «*Fuzzy Analogy*: A new Approach for Software Cost Estimation». *Proceedings of the 11th International Workshop on Software Measurements*, août, Montréal, p. 93-101.

Idri, Ali, Taghi Khoshgoftaar et Alain Abran. 2002a. «Can Neural Networks be easily Interpreted in Software Cost Estimation?». *Fuzz-IEEE*, mai, Hawai, p. 1162-1166.

Idri, Ali, Alain Abran, Serge Robert et Taghi Khoshgoftaar. 2002b. «Estimating Software Project Effort by Analogy based on Linguistic Values». *8th International Symposium on Software Metrics*, IEEE computer Society, juin, Ottawa, p.21-30.

Idri, Ali, Samir Mbarki et Alain Abran. 2002c. «Interprétation des réseaux de neurones en estimation du coût de logiciels». *CARI*, Octobre, Cameroun, p. 221-228.

Idri, Ali, Taghi Khoshgoftaar et Alain Abran. 2002d. «Investigating Soft Computing in Case-Based Reasoning for Software Cost Estimation». *International Journal of Engineering Intelligent Systems*, vol. 10, no. 3, septembre. p. 147-157.

Idri, Ali, Alain Abran, Serge Robert et Taghi Khoshgoftaar. 2003. «Fuzzy Case-Based Reasoning Models for Software Cost Estimation». to appear in the *book Soft Computing in Software Engineering: Theory and Applications*, published by Springer-Verlag, 2003.

Jacquet, J.P., et Alain Abran. 1998. «Metrics Validation Proposals: A Structured Analysis». *8th International Workshop on Software Measurement*, septembre, Magdeburg, Germany.

Jager, Roger. 1995. «Fuzzy Logic in Control». Ph.D. Thesis, Technic University Delft, Pays-Bas.

Jang J. S. R. et C. T. Sun. 1992. «Functional equivalence between radial basis function networks and fuzzy systems». *IEEE Transactions on Neural Networks*. vol. 4, p. 156-158.

Jang, J. S. R., C. T. Sun et E. Mizutani. 1997. *Neuro-Fuzzy and Soft Computing*. Prentice-Hall.

Jordan, Michael, et I. Stuart Russell. 1999. «Computational Intelligence». *MIT Encyclopidia of the Cognitive Sciences*. Edited by Robert A. Wilson, Frank C. Keil, p. 73-90.

Jorgersen, M. 1995. «Experience with Accuracy of Software Maintenance Task Effort Prediction Models». *IEEE Transaction on Software Engineering*, vol. 21, no. 8, p. 674-681.

Kadoda, Ghada, M. Cartwright, L. Chen, et M. Shepperd. 2000. «Experiences Using Case-Based Reasoning to Predict Software Project Effort». *Proceedings of EASE*, , Keele, Grande Bretagne, p.23-28.

Kemerer, C.F. 1987 «An empirical Validation of Software Cost Estimation Models». *Communications of the ACM*, vol. 30, no. 5, p. 416-429.

Khoshgoftaar, Taghi M., et E. B. Allen. 1997. «Neural Networks for Software Quality Models». *Advances in Fuzzy Systems -Applications and Theory*: *Computational Intelligence in Software Engineering*, vol. 16, Editeurs: W. Pedrycz et J.F. Peters, p. 65-96.

Khoshgoftaar, Taghi M., M. P. Evett, E. B. Allen et P.D. Chien. 1997. «An Application of Genetic Programming to Software Quality Prediction». *Advances in Fuzzy Systems - Applications and Theory*: *Computational Intelligence in Software Engineering*, vol. 16, Editeurs: W. Pedrycz et J.F. Peters, p. 175-196.

Khoshgoftaar, Taghi. M., E. B. Allen et J. Deng. 2001. «Controlling Overfitting in Software Quality Models: Experiment with Regression Trees and Classification». *7th International Software Metrics Symposium*, IEEE Computer Society. p. 190-198.

Khoshgoftaar, Taghi M., B. Cukic, et N. Seliya. 2002. «Predicting Fault-Prone Modules in Embedded Systems Using Analogy-Based Classification Models». *International Journal of Software Engineering and Knowledge Engineering*: Special Volume on Embedded Software Engineering, vol. 12, no. 1, p.1-22.

Khoshgoftaar, Taghi. M., et N. Seliya. 2002. «Tree-Based software quality estimation models for fault prediction». *8th International Software Metrics Symposium*, IEEE Computer Society, p. 202-214.

Kitchenham, Barbara, S.L. Pfleeger et N. Fenton. 1995. «Towards a Framework for Software Measurement Validation». *IEEE Transaction on Software Engineering*, vol. 21, décembre. P. 929-944.

Kitchenham, Barbara, et S. Linkman. 1997. «Estimates, Uncertainty and Risks». *IEEE Software*, vol. 14, no. 3, p. 69-74.

Kitchenham, Barbara. 1998. «A Procedure for Analyzing Unbalanced Datasets». *IEEE Transaction on Software Engineering*, vol. 24, no. 4, p.278-301.

Kirsopp, C., M. Shepperd, et J. Hart. 2002. «Search Heuristics, Case-Based Reasoning and Software Project Effort Prediction». *Genetic and Evolutionary Computation Conference*, AAAI, juillet , Ney York.

Kohavi, R., et G. H. John. 1997. «Wrappers for Feature Selection for Machine Learning». *International Conference on Artificial Intelligence*, p. 273-324.

Kolodner, J. L. 1993. *Case-Based Reasoning*. Morgan Kaufmann.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge.

Krantz, D. H., R. D. Luce, P. Suppes, et A. Tversky. 1971. *Foundations of Measurement: Additive and Polynomial Representations*. Academic Press, vol. 1.

Kurtz, K. J., D. Gentner et V. Gunn. 1999. «Reasoning». Cognitive Science: Handbook of Perceptions and Cognition, pp145-200.

Leake, D. 1996. *CBR in context: the Present and the Future.* Case-Based Rasoning experiences, lessons and future directions, AAAI Press, Menlo Park.

Mac Cormac, R. R. 1985. *A Cognitive Theory of Metaphor*. MIT Press.

Marwane, R., A. Mili, 1991. «Building Tailor-made Software Cost Model: Intermediate TUCOMO», *Inf. Soft. Techn*., Vol. 33(3), April, p.232-238.

Matson, J., E. B. E. Barrett, J. M. Mellichamp. 1995. «Software Development Cost Estimation Using Function Points». *Transactions on Software Engineering*, vol. 20, no. 4, April, p. 275-287.

Mayhrauser, A. V. 1990. *Software Engineering: Methods and Management.* Academic Press, New York.

Medsker, L. R. 1994. *Hyprid Neural Networs and Expert Systems*. Kluwer Academic Publishers.

Miyazaki, Y., et K. Mori. 1985. «COCOMO'81 evaluation and tailoring». *Eighth Int. Conf. Soft. Eng.*, Londres, août, p.292-299.

Miyazaki, Y., M. Terakado, K. Ozaki et H. Nozaki. 1994. «Robust regression for developing software estimation models»". *Journal of software systems*, vol. 27, p.3-16.

Moody, J. et C. J. Darken. 1989. «Fast Learning in Networks of Locally-Tuned Processing Units». *Neural Computation*, 1, p. 281-294.

Morrison, D. F. 1983. *Applied Linear Statistical Models*. Prentice-Hall, Englewood- Cliffs, New Jersy.

Myrtveit, I., et E. Stensrud. 1999. «A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models». *IEEE Transactions on Software Engineering*, vol. 25, no. 4, juillet, p. 510-525.

Negnevitsky, M. 2001. *Artificial Intelligence: A guide to intelligent systems*, Addison-Wesley.

Newell, A., et H. A. Simon. 1972. *Human Problem Solving*. Englewood Cliffs, NJ, Prentice-Hall.

Newell, A. 1980. «Physical Symbol Systems». Cognitive Science, vol. 4, p.115-183.

Newell, A. 1990. *Unified Theory of Cognition*. Harvard University Press.

Niessink, F., et H. Van Vliet. 1997. «Predicting Maintenance Effort with Function Points». *Proceedings of the International Conference on Software Maintenance*, Bari, Italie, IEEE Computer Society.

Pal, S. K., T. S. Dillon et D. S. Yeung. 2001. *Soft Computing in Case-Based Reasoning*. Springer-Verlag.

Pedrycz, W., et J.F. Peters. 1997. «Computational Intelligence in Software Engineering». *IEEE Canadien Conf. On Electrical and Computer Engineering*, mai, p. 253-257.

Pedrycz, W., et Z. A. Sosnowski. 2001. «The Design of decision trees in the framework of granular data and their application to software quality models». *Fuzzy sets and Systms Journal*, vol. 123, p. 271-290.

Perlovsky, L. I. 2001. *Neural Networks and Intellect using Model-Based Concepts*. Oxford University Press.

Porter, A., et R. Selby. 1990. «Empiracally-Guided Software Development using Metric-based Classification Trees». *IEEE Software*, vol. 7, mars, p. 46-54.

Porter, A., et R. Selby. 1990. «Evaluating Techniques for Generating Metric-Based Classification Trees». *Journal System Software*, vol. 12, juillet, p. 345-361.

Proulx, Robert. 1994. «Le traitement symbolique et non symbolique dans les réseaux neuronaux: Un problème de catégorisation » *Lekton*, vol. 4, no. 2, p. 21-35.

Putnam, L. H. 1978. «A General Empirical Solution to the Macro Software Sizing and Estimation Problem». *Transactions on Software Engineering*, vol. 4, no. 4, juillet.

Pylyshyn, Zenon W. 1984. *Computation and Cognition: Towards a Foundation for Cognitive Science*. MIT Press.

Pylyshyn. Zenon W. 1989. «Computing in cognitive Science, Foundations of Cognitive Sciences». Éditeur: M. I. Posner, *MIT press*, p. 51-91.

Quinlan, R. J. 1986. « Induction on Decision Trees». *Machine Learning*, vol. 1, p. 81-106.

Quinlan, R. J. 1993. *Programs for Machine Learning*. San Mateo, CA: Morgan Kaufman.

Ramamoorthy, C. V., C. Chandra, S. Ishihara et Y. Ng. 1992. «Knowledge Based Tools Risk Assessment in Software Development and Reuse». *6th International conference on Tools with Artificial Intelligence*, p. 364-371.

Robert, Serge. 1995. «Intelligence artificielle et sciences cognitives». *Intentionnalité en Question*, Textes réunis et présentés par Dominique Janicaud, Librairie Philosophique J. Vrin, 231-250.

Rumelhart, David E., et J. McClelland. 1986. *Parralel Distributed Processing.* Explorations in the Microstructure of Cognition, vol. 1: Foundations, Cambridge, MIT Press.

Rumelhart, David E. 1989. «The Architecture of Mind: A Connectionist Approach». *Foundations of cognitive Sciences MIT press*, p.133-159.

Samson, B., D. Ellison et P. Dugard. 1993. «Software Cost Estimation using an Albus Perceptron». *8th International COCOMO Estimation meeting*, Pittsburgh.

Santini, S. et R. Jain. 1999. « Similarity Measures ». *IEEE Transactions on Pattern Analysis and Machine Intelligence*. vol. 21, no. 9.

Saporta Gilbert. 1990. *Probabilités , analyse des données et Statistique*. Editions Technip. Paris.

Schank, Roger. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computer and People*. Cambridge University Press,

Schofield, C. 1998. «Non-Algorithmic Effort Estimation Techniques». Tech. Report TR98-01, March. Bournemouth university, Grande Bretagne

Searle, John R. 1984. «Minds, Brains and Programs». *Behavioral and Brain Sciences*. vol. 3, p. 417-57.

Selby, R., et A. Porter. 1988. «Learning from exemples: Generation and evaluation of Decion Trees for Software Resource Analysis». *IEEE Transactions on Software Engineering*, vol. 14, p. 1743-1757.

Serluca, C. 1995. «An Investigation into Software Effort Estimation using a Backpropagation Neural Network». Mémoire de maîtrise, Bournemouth University, Grande-Bretagne.

Shepperd, Martin, C. Schofield, et B. Kitchenham. 1996. «Effort Estimation using Analogy». *Proceedings of the 18th International Conference on Software Engineering*, Berlin, p. 170-178.

Shepperd, Martin, et C. Schofield. 1997. «Estimating Software Project Effort Using Analogies». *Transactions on Software Engineering*, vol. 23, no. 12, November, p. 736-743.

Shepperd, Martin et G. Kadoda. 2001. «Using Simulation to Evaluate Predictions Systems». *Proceedings of the 7th International Symposium on Software Metrics*, avril, Grande-Bretagne, IEEE Computer society, p. 349-358.

Shukla K. K. 2000. «Neuro-Genetic Prediction of Software Development Effort». *Information Software Technology*, vol. 42, p. 701-713.

Skalak, D. B. 1994. «Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms». *11th International Machine Learning Conference*, Morgan Kauffmann.

Smolensky, P. 1988. «On the Proper Treatment of Connectionnism». *Behavioral and Brain Sciences*, vol. 11, 1988, p. 1-23.

Srinivasan, K., et D. Fisher.1995. «Machine Learning Approaches to Estimating Software Development Effort». *IEEE Transaction on Software Engineering*, vol. 21, no. 2, p. 126-136.

Stevens. S. S. 1946. «On the Theory of Scales and Measurement». *Science Journal*, vol. 103, p. 677-680.

Tirri, H. 1991. «Implementing Expert System Rule Conditions by Neural Networks». *New Generation Computing*, vol. 10, p. 55-71.

Tong-Tong J. R. 1995. *La logique floue*. Hermes, Paris.

Towell, G. G. 1992. «Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction». Ph.D dissertation, University of Wisconsin.

Troster J. et J. Tian. 1995. « Measurement and Decfect Modeling for a Legacy Software System ». *Annals of Software Engineering*. p. 95-118.

Turing, Alan M. 1936. «Théorie des nombres calculables, avec une application au problème de la décision », Traduit et annoté par Julien Bash., voir Girard et Turing, 1995, p. 49-104

Turing, Alan M. 1956. «Computing Machinery and Intelligence» Traduit par Patrice Blanchard, Voir Girard et Turing, 1995, p. 135-175.

Tversky, Amos. 1984. «Features of Similarity». *Psycological Review*, p.327-352.

Vicinanza, S. et M.J. Prietulla. 1990. «Case-Based Reasoning in Software Effort Estimation». *Proceedings of the 11th International Conference on Information Systems*.

Wittig, G. Finnie. 1997. «Estimating Software Development Effort with connectionist Models». *Information and Software Technology*, vol. 39, p. 469-476.

Yager, Ronald R. 1996. «Quantifier Guided Aggregation using OWA Operators». *International Journal of Intelligent Systems*, vol. 11, p.49-73.

Yager, Ronald R, et J. Kacprzyk. 1997. *The Ordered Weighted Averaging Operators: Theory and Applications*. Kluwer Academic Publishing, Norwell, MA.

Yager, Ronald R. 2000. «Fuzzy Constraint Satisfaction for E-commerce Agents». *7th International Conference on Fuzzy Theory and Technology*, Atlantic City, février, p. 111-114.

Zadeh, Lotfi A. 1965. «Fuzzy Set». *Information and Control*, vol. 8, p. 338-353.

--------- . 1971. «Similarity Relations and Fuzzy Ordering». *Information Sciences*, p.177-200.

---------. 1973. «Outline of a New Approach to the Analysis of Complex Systems and Decision Processes». *IEEE Transaction Syst. Man Cybern*, vol. 3, p. 28-44.

---------. 1975. «Calculus of Fuzzy Restrictions». *In Fuzzy sets and their applications to cognitive and decision Processes*, L. A. Zadeh, K. S. Fu, M. Shimura, Eds, Ney Work, p. 1-39.

---------. 1978. «PRUF- A Meaning Representation Language for Natural Languages». *Int. Journal. Man-Mach Studies*, Vol. 10, p. 395-460.

----------. 1979. «A Theory of Approximate Reasoning». *Machine Intelligence*, vol. 9, 1979, p. 149-194

----------. 1979. «Fuzzy Sets as a Basis for a Theory of Possibility». *Fuzzy Sets and Systems*, vol 1, 1979, p. 3-28.

----------. 1981. «Test-score Semantics for Natural Languages and Meaning Representation via PRUF». *Empirical Sematics*, Éditeur B. Reiger, p.149-194.

----------. 1983. «A Computational Approach to Fuzzy Quantifiers in Natural Languages». *Computing and Mathematics*, vol. 9. p. 149-184.

----------. 1986. «Outline of a Computational Approach to meaning and Knowledge Representation Based on a Concept of a Generalized Assignment Statement». *Proc. International Seminar Artificial Intelligence*, Man-Manch, Syst, Éditeurs: M. Thoma et A. Wyner, E, Springer-Verlag, p.198-211.

----------. 1994. «Fuzzy Logic. Neural Networks and Soft Computing». *Communications of ACM*, vol. 37, no. 3, p.77-84.

----------. 1996. «Fuzzy Logic = Computing with Words». *IEEE Transactions on Fuzzy Systems* , p. 103-111.

----------. 1997. «Some Reflections on the Relationship Between AI and Fuzzy Logic: A Heretical View». *IJCAI*, Springer, p.1-8.

----------. 2001. «From Computing with Numbers to Computing with Words - from Manipulation of Measurements to Manipulation of Perceptions». *Computing with words*, Éditeur: Paul P. Wang, John Wiley et Sons, p. 35-68.

Zimmerman, H. J. 1991. *Fuzzy Set Theory and Its Applications*. 2nd Edition Boston, MA: Kluwer.

Zuse, Horst. 1994. «Foundations of Validation: Prediction and Software Measures». *In Proceedings of the AOSW*, avril, Portland.

----------. 1998. *A Framework of Software Measurement.* de Gruyter.

----------. 1999. «Validation of Measures and Prediction Models». *Proceedings of the 9th International Workshop on Software Measurement*, septembre, Lac-Supérieur, Canada.