Université de Montréal

Identification des objets dans les applications léguées basée sur les algorithmes génétiques

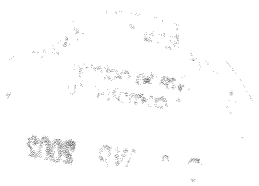
par

Idrissa KONKOBO

Département d'Informatique et de Recherche Opérationnelle
Faculté des Arts et des Sciences

Mémoire présenté à la Faculté des Études Supérieures
en vue de l'obtention du grade de
Maître ès Sciences (M.Sc.)
en Informatique

Décembre, 2001



Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Identification des objets dans les applications léguées basée sur les algorithmes génétiques

présenté par :

Idrissa KONKOBO

a été évalué par un jury composé des personnes suivantes :

Petko VALTCHEV
Président-rapporteur

Houari A. SAHRAOUI
Directeur de recherche

Jacques FERLAND
Membre du jury

Mémoire accepté le : 20 février 2002

Résumé

De nos jours, la maintenance est reconnue comme une activité incontournable dans le cycle de vie d'un logiciel; elle reste cependant très coûteuse en termes de temps et d'argent. Afin d'atténuer ces facteurs limitants, de nombreuses organisations informatiques optent pour une migration vers les technologies émergentes notamment celles orientées objets du fait des nombreuses possibilités qu'elles offrent.

Une multitude d'approches ont été définies pour résoudre le problème de la migration; ces dernières sont autant fonction des systèmes, des abstractions utilisées, que de leurs degrés d'automatisation. Dans ce travail, nous procédons à un inventaire et proposons une classification de ces approches en fonction des types d'applications auxquels elles sont destinées.

Nous présentons ensuite une approche globale de migration des systèmes procéduraux vers le paradigme orienté objet; nous insistons particulièrement sur l'identification des objets. Pour identifier les objets, nous utilisons une approche à la fois inspirée de l'intelligence artificielle, notamment les algorithmes génétiques, et des métriques de conception : les métriques de cohésion et de couplage. Pour les rendre applicables au problème d'identification des objets, des adaptations ont été apportées aux opérateurs génétiques classiques.

Le prototype que nous avons réalisé, a été testé sur trois systèmes de taille moyenne et les résultats obtenus sont satisfaisants. En outre, il est capable de fonctionner en mode automatique mais reste ouvert à l'intervention humaine notamment lorsqu'un expert du domaine est disponible pour ajuster les paramètres d'exécution. Enfin le prototype offre la possibilité de coopérer avec d'autres approches.

Mots-clés : systèmes légués, identification des objets, cohésion, problème de regroupement, couplage, croisement de regroupement, mutation de regroupement, retro ingénierie.

Abstract

Maintenance is considered to be an important activity in the software life cycles but it is also time and money consuming. To overcome these shortcomings, many computer organizations choose to migrate towards the emerging technologies, particularly to the object-oriented ones because of the numerous advantages that they offer.

Many approaches have been proposed aiming at solving the problem of the migration; theses approaches are based on the types of systems, on the used abstractions or on their automation degree. In this work, we studied the existing approaches thoroughly and made a classification of these approaches according to the type of applications.

Furthermore, we present a global approach of migration for the procedural systems towards the object-oriented paradigm. We focus on the objects identification step; to identify objects more effectively, we develop an approach which is a mixture of the genetic algorithms and some object oriented design metrics: cohesion and coupling. To make genetic algorithms applicable to the problem of identification of objects, some modifications were made on the classical genetic operators.

The prototype we built was tested on three medium sized systems and the results are very satisfactory. It can work in automatic fashion but it is also opened to human intervention when a domain expert is available to adjust the parameters of execution. Finally the prototype offers the possibility of cooperating with other approaches.

Keywords: legacy systems, objects identification, cohesion, coupling, grouping problems, grouping crossover, grouping mutation, reengineering.

TABLE DES MATIÈRES

Liste des tableaux

Liste des figures

Liste des abréviations

AG : Algorithmes Génétiques

AST : Abstract Syntax Tree (Arbre syntaxique abstrait)

CBO : Coupling Between Objects

ERCOLE : Encapsulation, Reengineering and Coexistence of Object with LEgacy

GELO : Laboratoire de GEnie LOgiciel

GNU : Gnu's Not Unix

GOAL : algorithme Génétique pour l'identification des Objets dans les Applications Léguées.

NP : Non Polynomial

OO : Orienté Objet

SDG : System Dependence Graph

À

f. Souleymane, mon père et

à Asséta « Guiba » ma mère bien aimée

Remerciements

L'occasion est pour moi, de remercier tous ceux qui ont, de près ou de loin, contribué à la réussite de mes études en général et de ce mémoire en particulier.

Un grand merci donc à M. Houari A. Sahraoui, mon Directeur de recherche pour sa disponibilité, ses conseils et ses soutiens de tous ordres.

Je tiens également à remercier le Programme Canadien de Bourses de la Francophonie (PCBF) qui a bien voulu m'accorder la bourse d'études qui m'a permise de suivre mon programme de maîtrise.

Je ne saurais terminer sans remercier tous mes amis et les membres du groupe de Génie Logiciel de l'Université de Montréal, particulièrement Shiqiang, Salah, Mohamed Adel, Hassan, Gang et Hong;

Que tous les membres de ma famille trouvent ici la reconnaissance des soutiens qu'ils m'ont toujours apportés.

Chapitre 1 Introduction

1.1 Contexte de ce travail

De nos jours, dans le domaine du génie logiciel, tout le monde s'accorde sur le fait que la maintenance logicielle prend une place très importante dans le cycle de vie des logiciels. En termes de coûts et de temps, la maintenance est demandeuse en ressources. En effet, les informaticiens passent de plus en plus leur temps à maintenir les logiciels. Dans [35], Pressman estime que 40 à 70 % du budget des organisations de développement de logiciel est consacré à la maintenance. Faciliter la maintenance de ces systèmes est donc crucial et peut s'avérer fort rentable. À cet effet, il est important de bien comprendre les composants de ces systèmes et les relations qui existent entre eux.

Malgré les importants budgets consacrés à la maintenance, les résultats ne sont pas toujours à la hauteur des attentes. Cette difficulté de maintenance est notamment liée au manque de documentation ou à l'existence d'une documentation qui n'est pas toujours à jour. Dans le but de résoudre ces problèmes, de nombreuses organisations tentent de faire migrer leurs systèmes vers des technologies émergentes possédant des qualités intrinsèques de maintenabilité. Au nombre de ces technologies, on peut citer le paradigme orienté objet. En effet, par la nature des abstractions (les objets) qu'elles définissent, les représentations orientées

objets sont plus proches de la réalité et de la perception humaine. En outre, le principe d'encapsulation permet de limiter la complexité des maintenances futures. L'approche objet offre en plus des mécanismes qui favorisent la réutilisation et l'évolution des systèmes.

Au vu de ces qualités, la technologie orientée objet s'impose donc comme une alternative crédible vers laquelle on peut faire migrer les systèmes légués[1]. Cette migration qui suppose l'identification des caractéristiques objets présentes dans les programmes légués (surtout ceux écrits selon le paradigme procédural) est selon Canfora dans [4], de plus en plus digne d'intérêt et est à même d'aider à la compréhension de la conception des systèmes. Liu et Wilde dans [31], ajoutent que l'identification des objets facilite la connaissance précise des données présentes dans les systèmes et la façon dont ces données sont créées, modifiées et comment elles interagissent.

La migration peut se faire de plusieurs façons; la première méthode est de redévelopper le système du début à la fin. Cette solution est très coûteuse en temps et en argent. De plus, comme le souligne De Lucia dans [10], elle n'offre pas de garantie de résultats, compte tenu du risque de dégradation de la conception originelle. En effet, les systèmes légués abritent très souvent des informations et des règles d'affaires qui risquent d'être perdues.

La deuxième alternative est d'utiliser un outil ou un ensemble d'outils pour dériver de façon semi-automatique un système à objets à partir des systèmes légués. Nous nous inscrivons dans ce deuxième courant.

La migration a donné lieu à plusieurs techniques. Certaines d'entre elles proposant des processus globaux de migration des systèmes légués et d'autres présentant au contraire des algorithmes ou des techniques d'identification des objets dans le code procédural.

[1] Les systèmes légués désignent les systèmes hérités et devenus vétustes. (Voir terminologie anglaise de *Legacy Systems*)

L'identification des caractéristiques objets (classes, objets, méthodes, ...) dans les programmes légués a également donné lieu à de nombreuses techniques et approches. Ainsi, nous pouvons distinguer d'une part, les outils qui dépendent du domaine et de l'environnement de développement; ils nécessitent des connaissances sur le domaine pour effectuer la migration. D'autre part, on a les outils dits indépendants du domaine. Comme le dit Sahraoui dans [37], ces outils n'utilisent que le code comme entrée et des heuristiques pour prendre les décisions lors de l'identification des objets.

Dans ce mémoire, nous présenterons notre approche pour l'identification des objets. Elle s'inspire largement de la technique des algorithmes génétiques et s'appuie sur les métriques et des propriétés désirables de conception. Elle s'inscrit par ailleurs, comme une étape du processus global de migration défini dans [37] par Sahraoui. Ce processus sera présenté en détail au chapitre 3.

1.2 Les contributions principales

Les contributions principales de ce travail peuvent se résumer dans les points suivants :

1. La proposition d'une nouvelle alternative à un problème réputé complexe car d'autres méthodes (approches basées sur les données, sur les types, sur la classification conceptuelle, ...) n'ont pas donné de résultats très satisfaisants.

2. L'introduction des critères de qualité de logiciel (couplage et cohésion) dans le processus d'identification des objets.

3. La mise en évidence de la capacité des algorithmes génétiques à résoudre le problème d'identification des objets.

4. L'identification et la détermination de l'importance relative des informations les plus pertinentes dans une application léguée concernée par un processus de migration.

1.3 Structure du mémoire

Ce mémoire est organisé comme suit.
Le chapitre 2 procède à un catalogage et une analyse des principales approches de migration et/ou d'identification d'objets dans les applications léguées.

Le chapitre 3 décrit dans un premier temps, l'approche globale de migration dans laquelle s'inscrit notre algorithme d'identification d'objets. Dans un second temps, il procède à l'énoncé formel du problème.

Le chapitre 4 présente de façon détaillée la solution que nous proposons. Ainsi, les nécessaires adaptations des opérateurs génétiques proposées par E. Falkenauer dans [12], y sont présentées.

Le chapitre 5 est consacré à la description du prototype qui implante la solution, à la présentation des résultats et à leur analyse.

Enfin le chapitre 6 résume brièvement le travail, les contributions majeures et dégage les pistes futures de recherche.

Chapitre 2 État de l'art

Dans [48], J.A. Zimmer affirme que la migration des systèmes légués vers des architectures orientées objets n'est pas une activité nouvelle. En effet, Lehman et Belady présentaient la migration comme un choix économique dans leurs trois lois sur l'évolution des grands systèmes publiés dans [30]. Par ailleurs, Jacobson et Lindström, sans suggérer de méthodes pour l'identification d'objets dans les systèmes légués, décrivent dans [25] différents scénarios pour les activités d'ingénierie et de réingénierie des systèmes légués vers les architectures orientées objets.

Les objets représentent une abstraction d'un état et les opérations qui agissent sur cet état. Dans leur étude sur l'évaluation de la modularité des systèmes dans [5], Canfora et son équipe, présentent l'identification des objets dans les systèmes légués comme un cas particulier de décomposition/recomposition des composants d'un système légué en modules.

Plusieurs techniques d'identification ont été définies. Leur classification peut être faite selon plusieurs critères (le degré d'automatisation, le type d'information manipulée, ...). Lakhotia a proposé un *framework* de classification basé sur les

graphes[29]. Récemment, Koschke dans [28], a pour sa part proposé une classification en techniques automatiques et techniques semi-automatiques.

Dans la présentation de l'état de l'art, nous exposerons l'éventail des travaux qui ont été faits dans le domaine de la migration des systèmes légués d'une manière générale, mais notre intérêt sera porté particulièrement sur ceux réalisés dans le cadre de l'identification des objets. Ainsi nous présenterons d'abord les travaux réalisés pour la migration des applications procédurales (C, Pascal, Fortran, ...) puis suivront ceux réalisés dans le cadre de la migration des applications dédiées à la manipulation des données (Cobol, RPG, ...).
Quelque soit le cadre de migration, certaines abstractions sont souvent utilisées; avant de présenter l'état de l'art en matière de migration de systèmes légués, nous nous attacherons à présenter succinctement quelques unes de ces abstractions.

2.0 Les abstractions utilisées

De nombreuses approches de migration utilisent les mêmes types d'abstractions des programmes à migrer; au nombre de ces modes de représentations, les graphes de références, d'interdépendances et les treillis de Galois sont très souvent utilisés. Nous donnerons dans cette section, un aperçu de ces abstractions.

2.0.1. Les graphes de références

Dans les graphes de références, les nœuds sont soit des routines soit des variables globales; dans ce type de graphe, l'existence d'un lien entre une routine *r* et une variable *v* signifie que la routine *r* utilise la variable *v*.
Parmi les approches exploitant les graphes de références, on peut citer entre autres celle de Dunn et Knight[11] présentée à la section 2.1.2, celle de Canfora[4] présentée à section 2.1.3 et celle de Sahraoui dans [37] (section 2.1.4).
La figure 1a présente un exemple de graphe de références construit à partir du tableau I qui montre les relations entre les routines ri et les données di d'un programme; les ti représentent les types des données globales.

	d1(t1)	d2(t1)	d3(t2)	d4(t3)	d5(t3)	d6(t4)
r1			1			
r2	1		1			
r3		1				
r4		1				
r5	1					1
r6					1	
r7	1					1
r8		1		1		
r9				1		
r10					1	
r11	1		1			

Tableau I. Relations entre routines et variables globales.

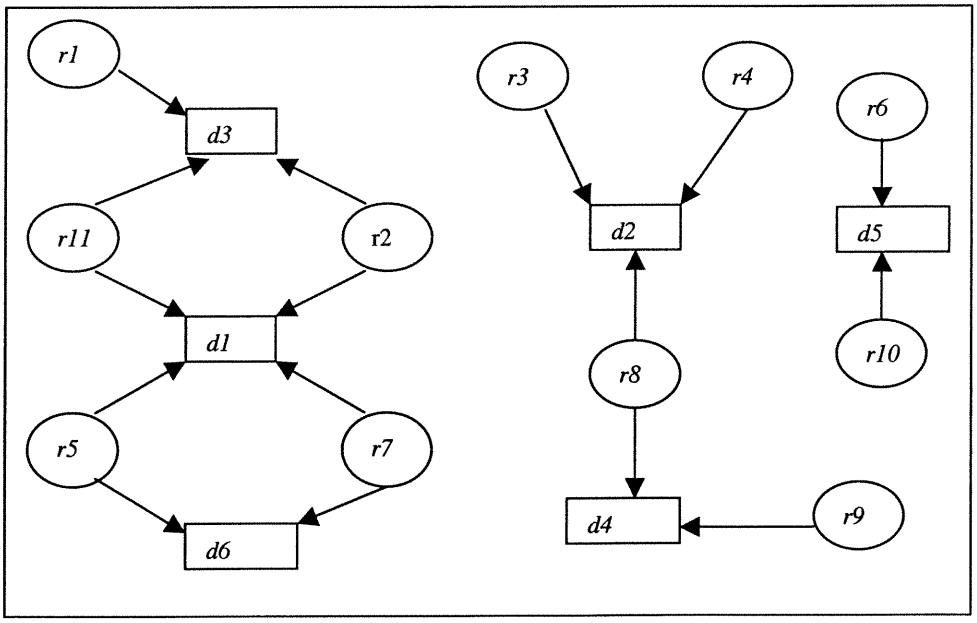


Figure 1a. Graphe de références des relations du tableau I.

2.0.2. Les graphes d'interdépendances

Dans un graphe d'interdépendances entre routines les nœuds sont des ensembles de routines; un nœud représente l'ensemble des routines qui font référence à une même donnée. Dans un graphe de ce type, l'existence d'un arc entre deux nœuds signifie que les deux sous ensembles que ces nœuds représentent sont non disjoints; c'est-à-dire qu'il existe une routine r dans les deux nœuds, qui fait référence à deux données d1 et d2.
Les graphes d'interdépendances ont été utilisés par Liu et Wilde dans [31] (voir section 2.1.1).

Un exemple de graphe d'interdépendances extrait du tableau I est donné à la figure 1b.

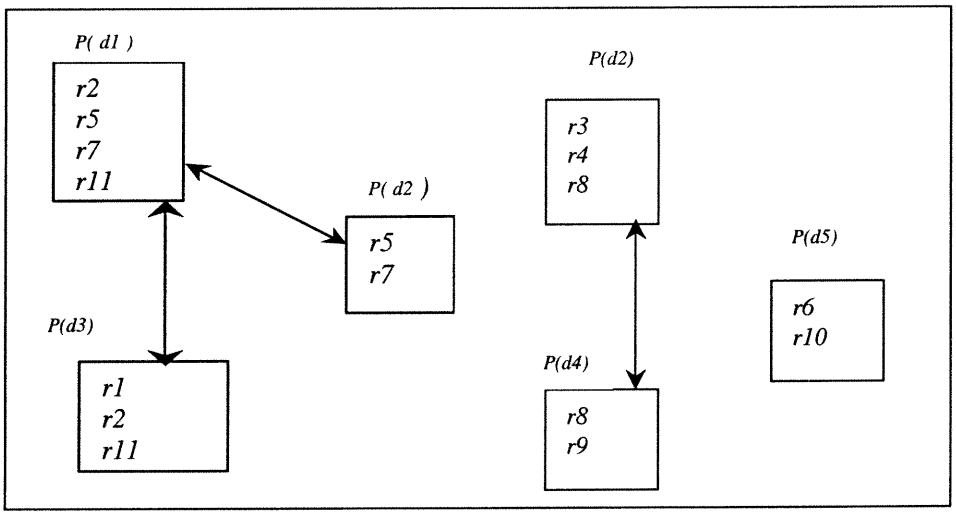


Figure 1b. Graphe d'interdépendances des relations du tableau I.

2.0.3. Les treillis de Galois

Cette section présente une définition succincte des treillis de Galois; pour une description plus approfondie voir les travaux de [3], [8] et de [47].
Selon [18], la notion de treillis de Galois (ou treillis de concepts) est à la base d'un grand nombre de méthodes de classification conceptuelle; [37] les utilise dans son approche d'identification des objets.

Définition
Soient deux ensembles finis, E et E', et *R* une relation binaire définit entre ces deux ensembles.
Soient P(E) et P(E') les ensembles des sous ensembles de E et E' respectivement.
Soit $\perp$ une relation binaire définie entre des paires de sous ensembles (X, X'), avec $X \subseteq E$ et $X' \subseteq E'$, tel que $(X,X') \perp (Y,Y')$ si et seulement si on a $X \subseteq Y$ et $Y' \subseteq X'$.
Soit (X,X') une paire d'ensembles. On dit que (X,X') est complet par rapport à la relation *R* si et seulement si :

1. X' est l'ensemble des images communes des éléments de X par la relation *R*, c'est-à-dire $X' = f(X) = \{x' \in E' \mid \forall x \in X,\ x\ R\ x'\}$
2. X est l'ensemble des antécédents communs des éléments de X' par la relation *R*, c'est-à-dire $X = f'(X') = \{x \in E \mid \forall x' \in X', x\ R\ x'\}$

Le treillis de Galois est l'ensemble des éléments (X, X'), où $X' \in P(E)$ et $X \in P(E')$

Des techniques basées sur les treillis de Galois ont été utilisées entre autre par Sahraoui dans [37] (section 2.1.4) et par Van Deursen et Kuipers dans [46] (section 2.2.1).

La figure 2a (tirée de [37]) donne un exemple de treillis de Galois construit à partir de la relation binaire de la figure 2b.

R	a	b	c	d	e	f	g	h	i
1	1		1			1		1	
2	1		1				1		1
3	1			1			1		1
4		1	1			1		1	
5		1			1		1		

Figure 2b. Représentation d'une relation binaire R.

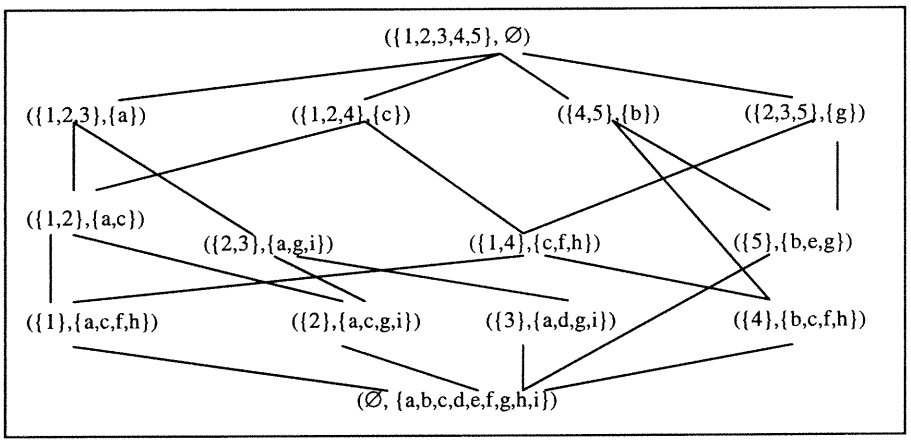


Figure 2a. Treillis de Galois de la relation R de la figure 2b.

Après avoir présenté les abstractions utilisées, dans la section qui va suivre nous allons présenter l'état de l'art en matière de techniques de migration des applications léguées. Comme nous l'avons dit auparavent, ces techniques se regroupent en techniques pour la migration des applications écrites avec les langages procéduraux et en techniques pour la migration des applications à forte manipulation de données.

2.1 Les techniques de migration des applications écrites avec les langages procéduraux

2.1.1 L'approche de Liu et Wilde

Dans l'outil *Object Finder[31]*, Lui et Wilde, après avoir fait l'inventaire des variables et des routines, définissent un graphe d'interdépendances entre ces variables et ces routines.
Liu et Wilde remarquent que les sous graphes représentent des objets-candidats. Cependant, dans les graphes d'interdépendance de routines il peut apparaître des liens indésirables entre des objets. Par exemple, si on a une routine *r* qui sert à initialiser plusieurs objets, le graphe fera apparaître un lien indésirable entre cette routine et les objets *O1* et *O2* faisant ainsi penser que ces 3 entités (routine, objet *O1*, objet *O2*) forment un seul objet alors que cette conclusion est erronée. Malheureusement, l'approche de Liu et Wilde ne résout pas ce genre de problème.

De plus, aucun élément dans l'algorithme de Liu et Wilde, ne permet de distinguer les routines qui implémentent les méthodes du même objet (et qui doivent être regroupées avec la donnée globale à laquelle elles accèdent) et les routines accédant à des données globales qui concernent plusieurs objets.

On peut noter aussi que dans l'approche Liu et Wilde, dans le cas où le couplage entre les composants du programme est élevé, un seul graphe avec une valeur élevée de couplage apparaît comme objet candidat. Ce problème est résolu par l'utilisateur qui doit spécifier (à la main) les variables globales qui sont responsables de ces liens non désirables. Ainsi, durant la phase d'analyse ces variables globales seront alors ignorées faisant du même coup disparaître ces liens.

2.1.2 L'approche de Dunn et Knight

Elle est basée sur les graphes de références présentés à la section 2.0.a. Comme dans [4], Dunn et Knight aboutissent aussi à la conclusion que chaque sous graphe isolé identifie un objet. De plus, dans [11] Dunn et Knight utilisent un système expert pour prendre les décisions afin de favoriser la création d'objets les plus réutilisables possibles. La base de connaissances de leur système expert est composée des règles de réingénierie, de métriques et de règles de conception.

Dunn et Knight font partie des pionniers en matière d'identification des objets. En effet, leur approche est l'une des premières solutions proposées pour l'identification des objets. Cependant, elle apparaît comme idéaliste dans la mesure où elle s'appuie sur les sous graphes isolés alors que dans les applications léguées, il n'est pas toujours évident de trouver les données et les routines regroupées de façon aussi parfaite.

2.1.3 L'approche de Canfora

Dans [4], Canfora et son équipe proposent un algorithme qui transforme un graphe de références en un ensemble de sous graphes fortement connectés et disjoints. Pour résoudre les problèmes d'existence de liens non désirés entre objets, ils calculent aussi pour chaque sous graphe g un index de connexion interne noté ICg; cette valeur mesure le couplage d'un groupe par rapport au reste du système.

L'algorithme procède de la manière suivante: à chaque étape de l'algorithme on associe à chaque routine r un ICg; ce dernier mesure l'utilisation de r pour générer un groupe g. Le groupe g inclut l'ensemble des données référencées par r et toutes les autres routines auxquelles cet ensemble de données fait également référence. La variation ΔIC représentant la différence entre la connectivité interne du sous graphe généré et les connectivités internes des autres sous graphes est aussi calculée.

Par la suite, une valeur seuil pour ΔIC est fixée; au-dessus de ce seuil, la variation de IC est considérée comme mauvaise et les routines associées sont considérées comme introduisant des connexions fausses ou de coïncidence.

À chaque étape une fonction de filtrage statistique est utilisée pour calculer une valeur seuil Δ, et:

* si $\Delta > \Delta IC$ on applique une fonction de fusion (*merging*) qui met dans un même groupe toutes les données des sous graphes.

* si $\Delta < \Delta IC$ on applique une fonction de découpage (*slicing*) qui découpe une routine dans le but de dissocier les deux sous graphes.

Bien que fournissant des résultats appréciables, l'approche de Canfora est difficilement automatisable du fait de la méthode de calcul du seuil d'étape. En effet, ce seuil est calculé en tenant compte du style de programmation, dont la détermination peut être subjective.

Dans [6], Canfora et son équipe proposent une variante de ΔIC qui va au-delà de la détection des objets. Elle permet en effet, de détecter des types de données abstraits qui s'avèrent être très utiles dans un processus de retro ingénierie. Leur contribution majeure est qu'à la différence des autres méthodes de regroupement utilisant ΔIC, cette méthode utilise des métriques de cohésion et de couplage pour générer les groupes.

2.1.4 L'approche de Sahraoui

La démarche présentée par Sahraoui et son équipe dans [37], s'inscrit dans une approche globale de migration des programmes procéduraux vers les technologies à objets. Ils définissent un paradigme en 5 étapes:

- caractérisation et extraction des informations
- identification des objets
- identification des méthodes

- identification des relations
- transformation du code

Ces différentes étapes seront présentées en détail dans le chapitre 3.
Dans l'étape d'identification des objets, les auteurs exploitent aussi le principe des graphes de références mais avec un algorithme basé sur une technique d'intelligence artificielle. En effet, leur approche s'appuie sur un concept bien connu dans ce domaine: la formation de concept. Dans [16], Girard définit un concept comme un ensemble d'entités (généralement des fonctions) et un ensemble d'attributs (variables référencées, types de signature,...) dans lequel chaque entité possède tous les attributs.

À partir d'un graphe de références, Sahraoui et son équipe définissent une matrice M où les lignes représentent les variables et les colonnes les procédures. Une entrée $M_{l,c}$ est soit égale à 1 si la variable associée à la ligne *l* est utilisée dans la procédure associée à la colonne *c,* soit égale à 0 si la variable n'est pas utilisée.

Partant de cette matrice, le treillis de Galois correspondant est construit en se basant sur des heuristiques (regroupement des données ayant le même comportement, tailles des objets, ...). Dans ce treillis, les nœuds représentent des objets-candidats.
Une étape d'identification des objets finaux est ensuite appliquée et les objets-candidats sont soit fusionnés ou laissés en l'état. La fusion ou non des objets-candidats est basée sur le nombre de données qu'ils ont commun; plus ce nombre est élevé, plus on aura à les fusionner.

Un des avantages à relever dans cet algorithme est qu'il peut être appliqué automatiquement ou non (dans ce cas, les objets identifiés au cours du processus sont validés par un expert). Comme la plupart des techniques, lors de l'identification des objets dans les systèmes à forte composantes de bibliothèques, cette approche ne fait pas de différence entre les composants dépendants du domaine d'application et ceux indépendants du domaine (par exemple les

bibliothèques). Ce problème relève plus de la nature de ces types de système que des approches de migration.

2.1.5 L'approche de Panas et Johnson

Dans [34], Panas et Johnson présentent une approche pour la migration des programmes écrits en C et en Pascal. Cette approche procède en deux étapes : une étape d'identification des objets primaires et une autre d'identification des objets secondaires.

- **identification des objets primaires**

Elle permet d'identifier les « objets évidents » présents dans le programme procédural. Panas et Johnson utilisent au cours de cette étape une méthode dite du « type du paramètre-récepteur ». Ils définissent le paramètre-récepteur d'une routine r comme tout paramètre de r qui est modifié au moins une fois au cours de l'exécution de la routine r. La méthode d'identification des objets basée sur le type du paramètre récepteur, regroupe une routine avec les types de ses paramètres. On peut remarquer qu'avec cette méthode, une routine pourrait être appelée avec plusieurs paramètres mais l'objet identifié et associé à la routine ne contiendrait que quelques uns de ces paramètres.

- **identification des objets secondaires**

Cette étape a pour but de compléter l'étape précédente; Panas et Johnson étendent leur approche pour résoudre les limites inhérentes aux approches automatiques. Ils définissent pour cela un ensemble de requêtes permettant de trouver des objets secondaires à partir d'objets primaires identifiés dans la phase précédente. Les types de requêtes proposées sont: la sélection, l'union, l'intersection, la soustraction et la suppression.

Avec cette méthode en deux étapes, Panas et Johnson montrent que les objets identifiés sont de bien meilleure qualité par rapport à ceux identifiés par les

méthodes basées uniquement sur les types. En effet, cette méthode arrive à déterminer des objets de granularité plus fine. De plus, elle permet de prendre en compte les procédures dont le corps est inconnu (par exemple les bibliothèques de procédures ou de fonction).

Les critiques que nous pouvons faire à l'encontre de cette technique sont de deux ordres; d'une part, on a la limite de la généralisation de la technique à d'autres langages. En effet, dans [34] les auteurs mentionnent que cette méthode n'est applicable que pour les programmes écrits en C et en Pascal;
En outre, cette méthode ne traite que les programmes écrits en C et en Pascal qui ne contiennent pas de variables pointeurs; dans le cas particulier des applications écrites en C, ceci rend la méthode très irréaliste dans la mesure où dans la réalité il est quasiment impossible d'imaginer des applications écrites en C qui ne contiennent pas de pointeurs.

2.1.6 L'approche de Girard

Girard et son équipe ont proposé dans [16], une approche appelée regroupement par similarité pour identifier des objets (encapsulations d'états abstraits) et des classes (types de données abstraits). Cette approche est basée sur des métriques de similarité.

Inspirée du travail de Schwanke[39], le calcul des métriques porte sur les entités élémentaires des programmes (routines, types et variables), mais prend aussi en considération des informations informelles telles que les commentaires, les noms des fonctions, etc. L'algorithme procède de façon très simple :

1. Mettre chaque entité dans un groupe
2. Répéter les étapes 3 et 4 jusqu'à ce que les groupes soient de « qualité satisfaisante »
3. Identifier les groupes les plus similaires
4. Les combiner

5. Supprimer les « groupes invalides »

La métrique de similarité est construite à partir de trois sous métriques de similarité et d'un facteur qui permet d'ajuster l'influence de chacune de ces sous métriques. Si on considère deux entités A et B, la métrique est calculée à partir des:

- relations directes : représentant les relations directes entre les deux entités A et B;
- relations indirectes : représentant les relations entre chaque entité et une troisième entité commune C;
- informations informelles : représentant les informations sans significations sémantiques dans le langage de programmation mais utilisées par le programmeur pour communiquer avec d'autres personnes; par exemple les commentaires, les noms des identificateurs, ...

Testé sur des systèmes de grande taille (~ 38 KLOC), le prototype qui implante cette approche a fourni des résultats fort appréciables; Cependant, comme Girard le mentionne dans [16], cette approche identifie un fort taux d'objets de la catégorie « faux positif » c'est-à-dire un fort taux d'objets identifiés mais qui n'existent pas dans la réalité.

Malgré la qualité des techniques pour la migration des applications écrites à l'aide des langages procéduraux, il faut reconnaître qu'elles se retrouvent fort limitées dans le cadre de la migration des systèmes à forte intensité de données. En effet, généralement, ces systèmes sont écrits en Cobol, en PL1 ou en RPG alors que ces langages sont reconnus comme présentant des différences avec les langages procéduraux ; au nombre de ces différences, on peut citer le passage de paramètres, les règles de portée des variables, les structures de contrôles, ...

Nous allons dans la deuxième partie de l'état de l'art, présenter quelques approches qui ont été présentées dans le cadre des applications écrites avec les langages orientés manipulation de données.

2.2 Les techniques de migration des applications écrites avec les langages orientés manipulation de données

Les techniques pour la migration des systèmes de manipulation de données nécessitent souvent de posséder des connaissances sur le domaine. Cette caractéristique est due à la nature même des abstractions utilisées par les langages qui servent à leur implantation. Tout au long de cette section, nous présenterons quelques-unes des approches définies pour ce type de systèmes.

2.2.1 L'approche de Van Deursen et Kuipers

Van Deursen et Kuipers proposent une méthode pour rechercher des objets dans les systèmes à forte intensité de données. Dans ces systèmes, les enregistrements sont naturellement considérés comme les structures de base. Ainsi, Van Deursen et Kuipers présentent dans [46] des méthodes basées sur :

- l'analyse par regroupement
- l'analyse de concepts; l'utilisation de cette approche mathématique pour l'identification des objets est de plus en plus répandue (voir [34, 40, 41, 42]).

a. L'analyse par regroupement

Van Deursen et Kuipers utilisent l'algorithme d'agglomération hiérarchique (voir [27]) dans leur approche. Leur but est d'identifier dans un ensemble de données des groupes d'enregistrements liés fonctionnellement.

L'algorithme proposé procède de la manière suivante :

- Définition d'une matrice qui capturent les références des procédures aux variables (cette matrice est identique à celle utilisée dans l'approche de Sahraoui[37] présentée à la section 2.1.4). Dans cette matrice, chaque ligne peut être vue comme un vecteur dont le nombre de 1 est égal au nombre de procédures qui utilisent la variable.

- Calcul des distances entre les variables(vecteurs) et définition de la matrice de dissimilarité correspondante.

- Chaque élément est mis dans un groupe à part, puis en fonction de la similarité on reconstruit de nouveaux groupes (regroupements d'anciens groupes). Le processus se poursuit ainsi jusqu'à l'obtention d'un seul groupe. Tous les groupes intermédiaires peuvent être vus comme des branches d'un arbre. Les groupes solutions sont ceux identifiés en traçant une ligne horizontale à travers l'arbre à une hauteur spécifiée par l'utilisateur.

Les résultats obtenus par cette méthode sont satisfaisants mais deux problèmes demeurent :

- Si 2 enregistrements possèdent une clé identique tout en ayant les autres parties de leurs enregistrements respectifs disjoints, alors cette méthode identifiera 3 groupes: un contenant les champs du premier enregistrement sans la clé, l'autre contenant les champs du deuxième enregistrement sans la clé et un dernier constitué seulement de la clé.

- Quand 2 groupes sont équidistants d'un autre, aucun élément de décision ne permet de faire le choix de regroupement. Le choix arbitraire qui en résulte peut conduire à de mauvais résultats.

En conclusion, l'analyse par regroupement est utilisable sous certaines conditions, dont entre autres, que les champs à regrouper ne soient ni très nombreux, ni très utilisés.

b. L'analyse de concept

Van Deursen et Kuipers introduisent l'analyse basée sur les treillis de Galois pour faire surtout une comparaison avec l'analyse par regroupement. Le but de l'analyse de concept est le même que précédemment. Cependant à la différence de l'analyse par regroupement, l'analyse de concept ne crée pas un seul groupe optimal, mais construit tous les concepts possibles via un treillis ; de plus elle ne groupe pas simplement des items, mais construit plutôt des concepts; les concepts correspondent à un ensemble d'items partageant les mêmes caractéristiques.

En comparant les deux approches, il apparaît que dans l'analyse par regroupement, les groupes finaux sont fortement dépendants des décisions de regroupement faites en début de chaque analyse, alors que l'analyse de concept donne toutes les possibilités pertinentes de regroupements. De plus, l'analyse de concept traite bien les cas où des champs clés peuvent apparaître dans plusieurs enregistrements (comme clé primaire et comme clé étrangère) alors que l'analyse par regroupement ne le traite pas bien.

2.2.2 L'approche COREM

Proposée par Gall et Klösch dans [15], l'approche COREM définit un processus en quatre étapes. La première est la retro-conception[2] qui consiste en l'extraction du code d'un certain nombre d'informations de conception: diagrammes de flots de données, de structures,.... À partir de ces informations, un diagramme entité/relation est généré; ce dernier est ensuite transformé en un modèle de l'application orientée objet équivalente; les auteurs nomment ce modèle RooAM.

C'est dans la seconde étape de modélisation de l'application qu'est réalisé le processus de migration. Celui-ci consiste en la création d'un autre modèle orienté

[2] Voir terminologie anglaise de *design recovery*

objet (appelée FooAM) basé sur les exigences d'analyse du programme procédural. Le processus de migration de l'application orientée objet est fait par un expert du domaine.

Dans la troisième étape (*mapping* des objets), les éléments du RooAM sont *mappés* avec ceux du FooAM pour donner le modèle de l'application cible (appelé TARGETooAM). Le modèle de l'application cible est la résultante des deux modèles.

Une dernière étape, appelée adaptation du code source, termine le processus de transformation au niveau du code source; elle s'appuie sur les modèles obtenus aux étapes précédentes.

La principale critique qui limite l'utilisation de cette approche est qu'en plus du coût d'analyse élevé qu'elle suppose, elle reste tributaire de la documentation qui n'est malheureusement pas toujours disponible ou à jour.

2.2.3 L'approche de De Lucia

De Lucia et son équipe définissent dans [10], le paradigme ERCOLE pour faire migrer les programmes RPG vers des plates formes orientées objets. Le processus se fait en six étapes séquentielles incluant des étapes de retro ingénierie et de réingénierie. ERCOLE est essentiellement basé sur les données persistantes; à chaque étape, une «abstraction d'un modèle OO» est définie; dans ce modèle, les sous routines et les programmes de traitement par lots représentent des méthodes candidates. Les données et les méthodes sont alors combinées de manière à obtenir des valeurs de métriques optimales.

Pour l'étape de réingénierie, des techniques d'enveloppage sont utilisées. Elles permettent aux anciens systèmes de pouvoir coopérer avec les nouveaux permettant ainsi une migration sélective, incrémentielle et donc peu risquée.

Plus tard dans [9], pour l'association des méthodes aux objets, De Lucia et son équipe proposent une méthode qui optimise des métriques orientées objets. La métrique orientée objet utilisée est le couplage. Sa mesure est faite à partir d'un graphe bipartite représentant les accès (opérations d'entrée/sortie) des routines vers les données.

2.2.4 L'approche MOORE

Fergen et son équipe ont développé l'approche MOORE[13], un outil qui permet de transformer des programmes écrits en Cobol en des programmes écrits en Cobol orienté objet. Loin d'être un convertisseur automatique, cet outil propose à chaque étape, des changements que l'utilisateur accepte ou non le menant ainsi, de façon progressive vers des objets.

À chaque enregistrement, est associé une pondération indiquant le nombre de références à cet enregistrement. Les paragraphes Cobol utilisant ou modifiant les enregistrements sont considérés comme des bases pour l'identification des méthodes. Ces paragraphes possèdent aussi une pondération qui est fonction des champs présents dans le paragraphe. Pour réduire le nombre de classes, à chaque fois qu'une classe est trouvée, une mesure de similarité est effectuée pour déterminer si les classes existantes ne l'incluent pas déjà. De plus, MOORE offre un « dépôt de données » utilisé pour le stockage des classes et des méthodes qui ont été identifiées comme réutilisables.

Néanmoins, il faut signaler que la méthode MOORE reste un outil très dépendant de l'environnement de développement. En effet, il ne fonctionne que pour les programmes exclusivement écrits en Cobol85, écartant du même coup la plupart des applications basées sur les autres versions de Cobol ou les autres langages de manipulation de données.

De plus MOORE, n'offre aucune indication au sujet de la division des classes de grande taille en classes de petite ou de moyenne taille.

2.2.5 L'approche de Newcomb et Kottik

Comme la plupart des techniques pour la migration des applications dédiées à la manipulation des données, l'approche de Newcomb et Kottik [33] est basée sur les enregistrements. Après analyse du code, un arbre syntaxique abstrait est créé. L'arbre est décoré avec les propriétés sémantiques du programme pour donner un arbre syntaxique augmenté. À partir de cette étape, plusieurs types d'analyses (analyse de portée des variables, analyse d'utilisation, analyse d'unités de programmes, ...) sont effectués pour déterminer les objets et les méthodes potentiels.

Cette technique très guidée par les caractéristiques de Cobol est quasi automatique. Il faut cependant relever que le fait de se baser sur les caractéristiques d'un langage, rend les programmes orientés objets obtenus très proches de ce langage, en l'occurrence les objets sont très proches des programmes Cobol.

2.3 Synthèse

De l'analyse de l'état de l'art en matière d'approches d'identification d'objets, il ressort les observations suivantes :

- La majorité des approches sont semi-automatiques; ceci est pour beaucoup dû au fait qu'il reste difficile voire impossible d'automatiser complètement l'activité de conception même si certains des critères de son évaluation (notamment les métriques) sont facilement automatisables.
- Les méthodes préconisées pour les systèmes à manipulation de données sont difficilement applicables aux applications procédurales et vice versa.

Le tableau II de la page suivante, donne un récapitulatif des techniques présentées dans ce chapitre.

Domaines d'application	Auteurs et références	Degré d'automatisation	Base théorique	Forces	Limites
Systèmes procéduraux	Liu et Wilde [31]	Semi-automatique	Graphe de dépendances variables-routines	Simple et intuitif	Ne fait pas de distinction entre les routines liées à un objet précis et celles communes à plusieurs objets.
	Canfora et al.[4]	Semi-automatique	Transformation d'un graphe de références en sous graphes disjoints	Résout le problème de l'existence de liens indésirables entre deux sous graphes	Difficulté de calcul du seuil d'étape (Δ).
	Sahraoui et al.[37]	* Automatique *Semi automatique	Treillis de Galois	Couvre tout le processus de migration; Applicable à des grands systèmes (car l'algorithme est de complexité linéaire)	Processus pas encore totalement complété.
	Panas et al. [34]	* Automatique *Semi automatique	Graphe des dépendances du système	Les objets identifiés sont de granularité fine.	Méthode orientée vers un sous ensemble des programmes écrits en Pascal et en C (sans pointeurs).
	Dunn et Knight [11]	Automatique	Graphes de références + Systèmes experts	Méthode pionnière	Ne traite pas les cas de graphes correspondant à des programmes complexes

	Girard et al.[16]	Semi-automatique	Métrique de similarité	Basée sur les métriques de similarité de Schwanke; Prend en considération des données informelles (commentaires, ….)	Taux important d'objets identifiés mais inexistants dans la réalité.
Systèmes à manipulation de données	Van Deursen [46]	Semi-automatique	Analyse de concepts Analyse par regroupement	Méthode formelle; Applicable à des grands systèmes .	la construction de l'arbre est complexe (cas de l'analyse par regroupement)
	Gall et Klösch [15]	Semi-automatique	Diagramme entité/ relation	Incrémentale et intuitif (diagramme E/R)	Coût d'analyse élevé; Fortement dépendante de la documentation disponible lors de la migration.
	De Lucia et al. [9, 10]	Semi-automatique	Graphes	Méthode incrémentale; Basée sur des métriques OO.	Ne fonctionne que pour les programmes RPG.
	Fergen et al. [13]	Semi-automatique	Heuristiques-Cobol85	Guidée par l'utilisateur; « Entrepôt » pour stocker les classes identifiées.	Les objets identifiés sont de granularité trop grosse. Ne fonctionne que pour les programmes écrits en Cobol85.
	Newcomb et Kottik [33]	Automatique	Analyse d'un AST augmenté des attributs sémantiques des programmes	Méthode automatique	Les objets identifiés sont trop proches des abstractions Cobol de départ

Tableau II. Récapitulatif des approches de migration présentées

Chapitre 3

L'identification des objets dans la perspective des problèmes de regroupement

L'identification des objets comme nous l'avons vu dans l'état de l'art, représente très souvent une étape dans les processus de retro ingénierie. Ne faisant pas exception, notre algorithme pour l'identification des objets s'inscrit également dans un processus général de migration: le processus défini dans [37].

Avant de donner les détails de notre algorithme, nous présenterons d'abord dans cette rubrique le processus global de migration dans lequel il s'intègre. Dans un deuxième temps, nous montrerons en quoi un problème d'identification d'objets dans un programme procédural peut être considéré comme un problème classique de regroupement.

3.1 Une approche globale pour la migration des systèmes légués

Rappelons brièvement que l'identification des objets présentée dans cette étude, s'inscrit comme une étape du processus général de migration défini par Sahraoui[37]. Comme l'indique la figure 3, ce processus se divise en 5 étapes :

- caractérisation et extraction des informations
- identification des objets
- identification des méthodes
- identification des relations
- transformation du code

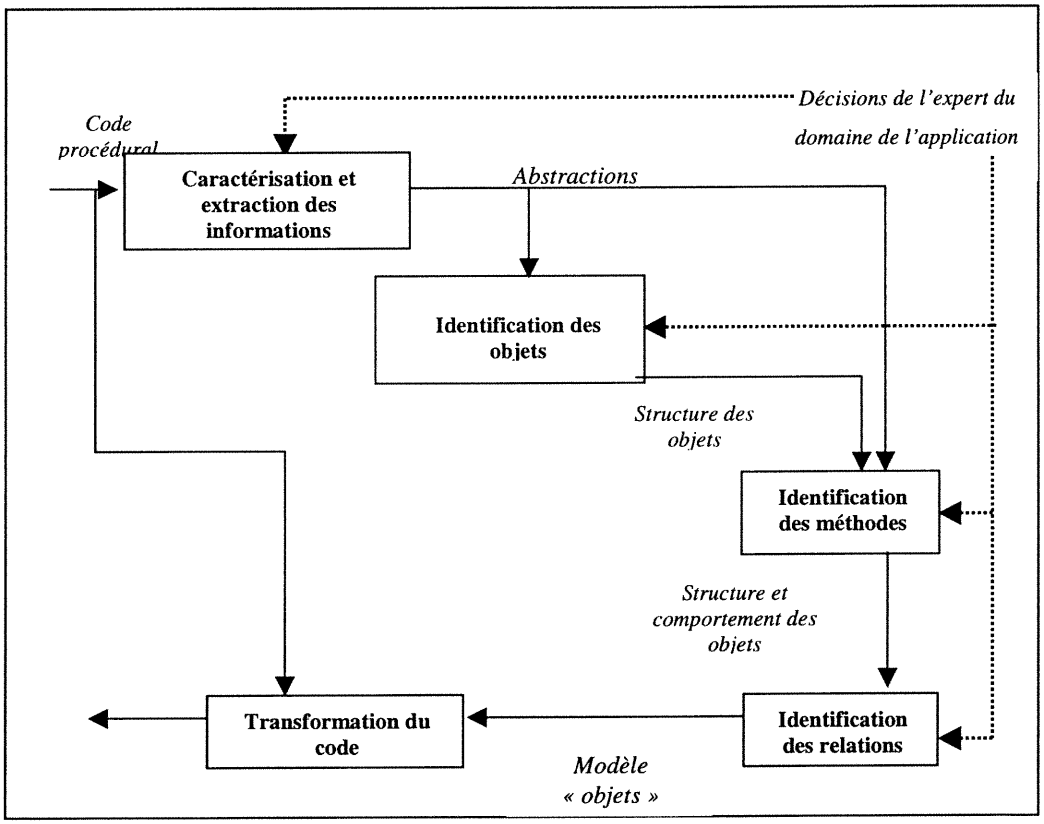


Figure 3. Approche globale de migration

Dans ce qui suit, nous allons présenter brièvement chacune des étapes précédemment citées.

3.1.1 La caractérisation des systèmes

Elle a pour but de déterminer les informations pertinentes à extraire pour assurer une migration réussie. Même si de nombreuses applications ont des structures similaires, il faut convenir avec [5] et [46] qu'en réalité les systèmes légués diffèrent sur plusieurs plans : langages de programmation utilisés, autonomie et nature des applications (librairies de fonctions ou non, systèmes à calculs intensifs,...). Par exemple si l'application à migrer est une librairie de routines, il pourrait être judicieux de ne pas se contenter des variables globales compte tenu du fait que dans ce type d'application les variables globales sont très peu utilisées.

3.1.2 L'identification des objets

L'identification des objets est l'étape la plus cruciale de la migration; elle constitue la matière principale du présent mémoire. Dans notre approche, cette identification est inspirée du principe qui consiste à regrouper les données du système en fonction des routines qui les manipulent. Pour faire le regroupement, nous utiliserons la technique des algorithmes génétiques couplée à des métriques de qualité de conception. Des détails seront donnés dans la section 3.2.

En utilisant le même paradigme de migration d'autres techniques ont été appliquées à cette étape. Ainsi, on peut citer l'approche de formation de concepts basée sur les treillis de Gallois[37] et l'approche basée sur l'algorithme de classification conceptuelle[41].

3.1.3 L'identification des méthodes

L'identification des méthodes traite uniquement la structure. En effet, à ce stade du processus, le comportement n'est pas encore défini. Ainsi, une même routine du système légué peut utiliser des données attribuées à plus d'un objet.

L'identification de méthodes consiste donc à transformer les routines en méthodes pour les différents objets identifiés. Le principe de transformation est simple lorsqu'une routine accède uniquement aux données reliées à un objet unique. La routine devient tout simplement une méthode de cet objet.

La transformation est plus complexe quand une routine utilise plus d'un objet. Différentes techniques sont utilisées dans ce cas. La première consiste à utiliser des heuristiques pour éviter la décomposition. Par exemple, une routine qui utilise plusieurs objets mais n'en modifie qu'un seul se transforme en une méthode de cet objet. Il est clair dans ce cas, qu'il faut remplacer les références explicites aux données des autres objets par des appels de méthodes pour respecter le principe d'encapsulation.

Une deuxième méthode utilisée pour la transformation est la technique de partitionnement (*slicing*). Cette technique permet de décomposer une routine qui utilise plusieurs objets en une séquence de routines ayant une exécution équivalente (même effet). Chacune des routines doit utiliser uniquement un objet et peut donc se transformer en une méthode de cet objet.

3.1.4 La découverte des relations

La plupart des travaux existants se limitent à l'étape d'identification des objets. Le modèle à objets obtenu par identification exploite très peu les avantages offerts par le paradigme orienté objet (héritage, agrégation, ...). Un tel modèle ne peut être exploité en l'état. Il s'agit donc dans cette étape, de découvrir les relations de types généralisation, agrégation et association qui lient les objets. La découverte des relations est considérée à tort comme une étape triviale, ce qui est loin d'être le cas particulièrement pour les systèmes de grande taille. Des techniques d'optimisation de modèles à objets ont déjà été utilisées [17].

À l'étape actuelle, les cas des agrégations et des associations n'ont pas encore été étudiés bien qu'ils aient été abordés dans [32].

3.1.5 La transformation/génération du code.

Elle peut se faire de plusieurs façons; par exemple l'implémentation du nouveau système en utilisant l'ancien système peut se faire avec des techniques d'enveloppage (*wrapping*). Elle peut également se faire par transformation ou génération de code. La transformation est beaucoup plus aisée dans le cas des langages procéduraux qui ont été étendus au paradigme orienté objet (par exemple C et C++), mais elle demeure complexe pour les autres langages.
Il faut cependant mentionner que certains problèmes peuvent demeurer, notamment celui du respect de l'encapsulation quand une routine de l'ancien système est transformée en plusieurs méthodes de différents objets dans le nouveau système.

3.2 L'identification des objets

De façon générale, dans les programmes écrits selon le paradigme orienté objet on trouve des classes et différents types d'associations liant ces dernières. Ces classes sont définies par :

- des attributs qui représentent, en terme d'abstraction, l'état des objets
- des méthodes qui accèdent à ces attributs afin de changer éventuellement cet état.

Dans une bonne conception orientée objet, les classes doivent être fortement cohésives et le couplage entre elles doit être réduit au minimum possible. C'est pourquoi, le concepteur consciencieux cherchera à obtenir une combinaison

optimale de ces deux propriétés, c'est-à-dire qu'il cherchera à maximiser la cohésion et à minimiser le couplage.

Par ailleurs, dans les programmes procéduraux on retrouve des données et des routines qui accèdent à ces données. Généralement, entre ces données il peut exister des similitudes; par exemple les données peuvent être définies et/ou modifiées dans les mêmes morceaux de codes. Cependant, malgré ces similitudes, ces données ne sont pas toujours mises ensemble de manière à pouvoir représenter des agrégats plus significatifs. En raisonnant par analogie, et en partant de l'hypothèse selon laquelle: indépendamment du domaine, dans un programme structuré, on peut trouver des objets en se basant sur les propriétés désirables de conception notamment le couplage et la cohésion , alors l'identification d'objets dans le code procédural peut être vue comme une activité de formation de groupes. Chaque groupe est constitué de données et des routines accédant à ces dernières.

Le problème de formation de groupes peut donc être défini comme un problème classique de regroupement. La cohésion d'un module ou d'une classe se définie comme l'homogénéité à l'intérieur de celui-ci; elle représente par conséquent, le degré de liaison entre les différents constituants d'un module ou d'une classe. Au moment de la conception, l'idéal c'est d'avoir la cohésion la plus élevée possible. Le couplage se définit comme étant le degré de liaison entre modules ou entre classes. Pendant la conception de son application, le concepteur cherchera à minimiser le couplage entre les classes ou entre les modules.

Dépendamment du domaine ou du résultat recherché, plusieurs techniques de mesure du couplage et de la cohésion ont été définies. Pour ce qui nous concerne, nous utilisons les métriques définies par Chidamber et Kemerer dans [7] et celles définies par Briand et son équipe dans [2]. Dans le paragraphe 5.4, nous donnerons plus de détails sur ces métriques.

3.2.1 Les problèmes de regroupement

Généralement, dans les problèmes de regroupement l'objectif est de partitionner un ensemble E d'objets en une collection de sous ensembles d'objets e_i respectant des conditions de complétude et de cohérence:

$$\text{Complétude : } \cup e_i = E \qquad (1)$$

$$\text{Cohérence : } e_i \cap e_j = \varnothing \qquad (2)$$

En d'autres mots, ces problèmes peuvent être vus comme des problèmes dont le but est de partitionner l'ensemble E (c'est-à-dire de regrouper les éléments de l'ensemble d'objets E, en un ou plusieurs (au maximum Card(E)) entités distinctes). Pour être considérées comme des groupes valides, les entités formées doivent satisfaire des contraintes qui peuvent changer en fonction des problèmes. Ces contraintes peuvent être par exemple la taille des groupes, la nature des groupes, les relations d'appartenance, ...

Notre problème consiste principalement à trouver des regroupements de données-routines. Nous souhaitons que ces groupes maximisent une fonction objectif qui sera définie en fonction du couplage, de la cohésion tout en respectant les contraintes de complétude et de cohérence de notre problème (c'est-à-dire, ils doivent constituer une partition de l'ensemble des données du programme):

- L'union des groupes obtenus doit être égale à l'ensemble des données du programme.
- Les groupes doivent être deux à deux disjoints (un élément de données appartient à un et un seul objet).

3.2.2 Formulation du problème d'identification d'objets basé sur les regroupements

Soient :

D l'ensemble des données d'un programme.

R l'ensemble des routines du même programme.

Identifier les « objets » présents dans ce programme, revient à constituer des groupes à partir des données en se basant sur les routines qui les manipulent. Ces groupes doivent constituer des partitions P de D. La meilleure partition Pi sera une de celles qui maximisent la valeur de la fonction objectif et respectent les contraintes de complétude et de cohérence.

Illustration :

Soit un programme procédural;

Soit l'ensemble de données D = {a, b, c, d, e, f, g } représentant les données présentes dans ce programme.

Soient les partitions P1, P2, P3, P4 définies par:

P1 = { {a, b}, {c}, {d, e, f, g} }

P2 = { {a}, {b}, {c}, {e}, {f},{g}}

P3 = { {a, b, c, d, e, f, g} }

P4 = { {a, b, c}, {d, e, f, g} }

Ces partitions sont des solutions potentielles à un problème de regroupement, donc d'identification d'objets, impliquant les données de l'ensemble D. On vérifie aisément que les partitions sont cohérentes et complètes c'est-à-dire que les groupes constituant chaque partition Pi sont disjoints et leur union est égale à l'ensemble D.

Une partition Pi sera retenue comme solution si la valeur de la fonction objectif pour Pi est supérieure ou égale à celles pour les autres partitions Pj.

Notons F la fonction objectif qui doit être dépendante de la cohésion et du couplage pour bien refleter les objectifs visés. Ainsi, une partition Pi sera retenue comme solution si :

F (cohesion(Ei), couplage(Ei)) $\geq$ F (cohesion(Ej), couplage(Ej)) pour tout j.

3.2.3 Les algorithmes génétiques, une solution aux problèmes de regroupement

La plupart des problèmes de regroupement sont des problèmes de classe NP, c'est-à-dire non déterministes polynomiaux. Les algorithmes pour les résoudre requièrent des temps d'exécutions exponentiels du nombre de données du problème. Or dans notre situation où les systèmes logiciels peuvent atteindre des millions de lignes de lignes de code, le temps d'exécution sera d'autant plus grand que le nombre de données de notre problème sera élevé.

Dans la littérature, on retrouve deux grands groupes de techniques pour résoudre les problèmes de regroupement; d'une part on a les méthodes énumératives et d'autre part les méthodes heuristiques.

Les méthodes énumératives

Ces méthodes incluent les techniques de *Branch and Bound[40b]*, les algorithmes A* (et les autres algorithmes de recherche basés sur les graphes), la programmation dynamique[], etc. Elles offrent la garantie de trouver l'optimum global. Cependant, face aux problèmes NP, ces techniques ont le défaut de ne pouvoir générer une solution en temps de résolution réalistes.

Les méthodes heuristiques

À l'opposé des méthodes énumératives, les méthodes heuristiques (par exemple *Simulated Annealing*, Recherche Tabou) ne garantissent pas l'optimum global. En revanche, de par leur nature stochastique, elles sont proches des variantes d'algorithmes génétiques où seul un opérateur de mutation est utilisé. Cependant, dans un espace de recherche très vaste (comme c'est le cas dans les problèmes NP complets), une méthode principalement basée sur une recherche aléatoire peut trouver difficilement une bonne solution.

Face aux limites de ces deux types de techniques, les algorithmes génétiques de regroupement tels que définis par Falkenauer[12], représentent une alternative

crédible pour résoudre les problèmes de regroupement. En effet, étant donnée que l'ensemble des solutions constituant l'espace de recherche est très grand, on recherchera donc des solutions acceptables et pas forcément des solutions optimales; Ainsi, les algorithmes génétiques représentent une alternative intéressante car :

- leur convergence est assurée grâce notamment à l'opérateur de mutation;
- ils fournissent des solutions acceptables;
- ils parcourent assez rapidement l'espace de recherche grâce aux opérateurs de reproduction (croisement et mutation).

Nous utiliserons un algorithme génétique de regroupement[12] pour résoudre notre problème. Dans la prochaine section, nous présenterons le principe général des algorithmes génétiques, leur applicabilité à notre problème et une présentation détaillée de GOAL, notre algorithme.

Chapitre 4

Un algorithme d'identification basé sur les algorithmes génétiques

Comme nous l'avons présenté dans le chapitre 2, plusieurs solutions ont été proposées aux problèmes d'identification des objets dans le code procédural avec plus ou moins de succès. Dans le présent chapitre, nous allons, sur la base de notre algorithme, présenter une solution au problème d'identification des objets; par la suite, nous présenterons les adaptations nécessaires préconisées par Falkenauer pour ce type de problème[12].

4.1 Principes généraux des algorithmes génétiques

La métaphore qui sous-tend la théorie des algorithmes génétiques est celle de l'évolution naturelle. L'idée de base est de dériver des solutions nouvelles et potentiellement meilleures à un problème en partant d'un ensemble de solutions initiales. L'aptitude de chaque solution est mesurée par une fonction objectif. À chaque génération, l'algorithme sélectionne des paires de solutions (*chromosomes*) en utilisant une méthode de sélection qui accorde la priorité aux solutions les plus aptes.

Sur chaque paire de solutions, on applique deux opérateurs : le croisement et la mutation et ce, dépendamment de la probabilité associée à chaque opérateur.

Chaque paire de chromosomes sélectionnée produit une nouvelle paire qui est incorporée dans la nouvelle génération. L'algorithme s'arrête lorsqu'un critère de convergence est atteint ou si un nombre fixé de générations est atteint. Dépendamment des politiques de sélection, le chromosome le plus apte de chaque génération est automatiquement ajouté à la nouvelle génération de manière à assurer que le chromosome le plus apte de la dernière génération est la meilleure solution du problème.

Les algorithmes génétiques sont des méthodes de recherche « aveugles » reposant sur les mécanismes de sélection naturelle définis par Charles Darwin: la survie du plus robuste.

J. H. Holland est l'initiateur du domaine de recherche des algorithmes génétiques. En 1975, dans sa publication intitulée « *Adaptation in Natural and Artificial Systems* »[23], il formula les principes fondamentaux des algorithmes génétiques :

a. La capacité de représentations élémentaires, comme les chaînes de bits, à coder des structures complexes.
b. Le pouvoir des transformations élémentaires à améliorer de telles structures.

Plus tard, les travaux de David E. Goldberg[20] contribuèrent largement à les vulgariser. En outre, il a énormément enrichit la théorie des algorithmes génétiques. Il fit le parallèle suivant: un individu est lié à un environnement par son code d'ADN et une solution est liée à un problème par son indice de qualité et conclut qu'une « bonne solution » à un problème peut être vue comme un individu susceptible de survivre dans son environnement donc ayant un indice de qualité élevé.

La terminologie employée dans les algorithmes génétiques est empruntée à la génétique; ainsi on y retrouve entre autres, les termes suivants:

a. *chromosomes:* ce sont les éléments à partir desquels sont élaborés les solutions.

b. *génération:* c'est l'ensemble des *chromosomes*. Les *générations* sont également appelées des *populations*.

c. *reproduction:* c'est l'étape de combinaison des *chromosomes*. La mutation et le croisement génétiques sont des méthodes de *reproduction*.

À la précédente liste, il faut rajouter des termes propres au domaine des algorithmes génétiques :

a. *le fitness* ou *indice de qualité* ou encore *indice de performance*, est une mesure permettant de classer les chromosomes selon un certain ordre en tant que solution à un problème.
b. la *fonction d'évaluation* est la formule qui permet de calculer *l'indice de qualité* d'un *chromosome*.

4.2 GOAL : un algorithme Génétique pour l'identification des Objets dans les Applications Léguées

L'algorithme, nommé GOAL, que nous proposons suit le schéma général d'un algorithme génétique classique; la figure 4 de la page suivante, donne l'allure générale de l'algorithme. Cependant, comme nous le verrons au paragraphe 4.2.2, GOAL utilise une autre représentation chromosomique des individus et redéfinit les opérateurs de reproduction. En outre, notre algorithme peut utiliser comme population initiale des solutions obtenues après application d'un autre algorithme d'identification d'objets basé sur la classification conceptuelle (voir [41]) ou après une génération aléatoire.

GOAL utilise en entrée un ensemble de solutions; les solutions sont toutes caractérisées à partir d'une matrice qui capture un ensemble de relations du type : « la donnée d_j est utilisée par la routine r_i». Dans une telle matrice, si la donnée d_j est utilisée par la routine r_i, l'entrée i,j de la matrice sera marquée du nombre 1 sinon elle sera laissée vide. Un exemple de ce type de matrice est présenté dans la section 4.3

```
i = 0
Créer la génération initiale Gi
maxSize = Taille de la génération initiale
Evaluer la génération Gi
BestFit = le Meilleur Chromosome de Gi
TantQue not EndCondition[1] Faire
        Créer une nouvelle génération Gi+1
        Ajouter BestFit à la nouvelle génération
        TantQue (taille de la nouvelle génération < maxSize) Faire
                Sélectionner 2 chromosomes C1 et C2
                Croiser C1 et C2 pour obtenir les chromosomes fils C_Fils1
                et C_Fils2
                Muter C_Fils1
                Muter C_Fils2
                Ajouter C_Fils1 et C_Fils2 à la nouvelle génération
        FinFaire
        Evaluer les chromosomes de la nouvelle génération
        BestFit = le Meilleur Chromosome de Gi+1
        i++
FinFaire
Return BestFit;
```

[1] *EndCondition* : peut représenter le nombre maximum de générations souhaité ou la convergence de l'algorithme

Figure 4. Allure générale de l'algorithme

4.2.1 Principe

Du fait de la particularité du problème à résoudre, les principes des algorithmes génétiques classiques (codage binaire, opérateurs classiques de croisement et de mutation) ne peuvent être appliqués en l'état. En effet, nous devons effectuer certaines adaptations liées à notre problème. Ces adaptations touchent aussi bien les étapes de représentation des chromosomes que de définition des opérateurs génétiques.

4.2.2 La modélisation

De façon classique, les algorithmes génétiques utilisent souvent des codages binaires ou réels pour décrire les points de l'espace d'états (les chromosomes) sur lequel ils opèrent. Ces types de codage, notamment le codage binaire, ont pour conséquence d'avoir des opérateurs de croisement et de mutation simples et de permettre le développement de solutions robustes. C'est ainsi que les premiers résultats de convergence théorique ont été obtenus à l'aide des représentations binaires des chromosomes (Voir [1]).

Cependant, comme le mentionne Jean-Marc Alliot dans [1], ce codage présente quelques limitations dépendamment du type de problèmes; par exemple le codage binaire peut rapidement devenir mauvais pour des problèmes d'optimisation dans des espaces de grande dimension et aussi pour les problèmes de regroupement.

Dans notre problème, étant donné que la formation d'objets est guidée par une approche basée sur des données, dans chaque chromosome représentant une solution, les gènes seront constitués de données et des routines les manipulant. Les gènes vont représenter des objets-candidats. Ainsi donc la présence d'un ensemble de données dans un même gène signifiera que ces données sont des attributs de l'objet-candidat pour le problème.

Illustrons cela par un exemple.

Soit le problème d'identification d'objets dans lequel apparaît un ensemble de données D tel que D = { a, b, c, d, e, f, g}, extrait de l'exemple présenté en détail au paragraphe 4.3.

Comme stipulé dans la rubrique 3.2.2 (formulation du problème d'identification des objets), toute partition de D est une solution. Considérons par exemple, la partition Pi telle que: Pi = {{a, b}, {g, f}, {d, e, c}}

Si on considère les groupes suivants:

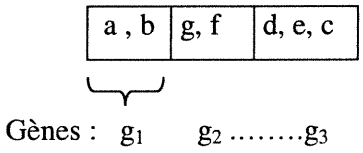
$g_1 = \{a, b\}$

$g_2 = \{g, f\}$

$g_3 = \{d, e, c\}$

alors Pi représente une solution probable car elle satisfait aux conditions de cohérence et de complétude (voir 3.2.1);

Cette solution est composée des « objets » O1, O2 et O3 représentés à la figure 5.
La représentation sous forme de chromosome de la solution représentée par Pi est :



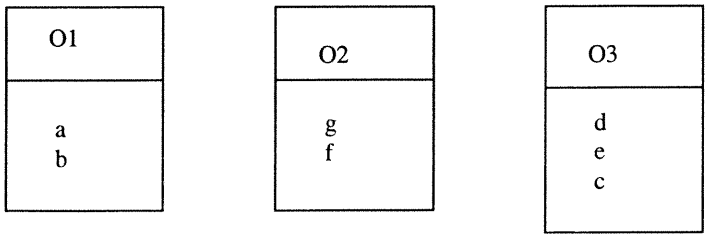


Figure 5. Représentation des objets O1, O2, O3

4.2.3 Génération de la population initiale

Le mécanisme de génération de la population doit être capable de produire une population non homogène d'individus; cette dernière servira de base pour la création des générations futures. Le choix de la population initiale est très important; en effet, en fonction du type d'initialisation utilisé, la convergence de l'algorithme vers l'optimum global peut être plus ou moins rapide.
Afin de ne pas biaiser les résultats, mais également pour conserver une bonne diversité, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.

Comme nous l'avons déjà signifié, pour créer la population de notre problème plusieurs possibilités s'offrent à nous:

a. l'approche aléatoire : à partir d'une méthode de génération de nombres aléatoires, des solutions sont constituées.

b. l'approche heuristique: la population initiale est obtenue après l'application de la première étape de l'algorithme de [41] qui constitue une extension de l'algorithme de Cobweb. Dans [41], Shen utilise le principe de la classification conceptuelle et son algorithme permet d'obtenir des groupes déjà significatifs au départ.

c. l'approche combinée : la population est constituée d'une combinaison d'individus générés aléatoirement et d'individus générés par l'algorithme de [41].

4.2.4 Les opérateurs génétiques

Nous utilisons les deux d'opérateurs les plus courants dans les algorithmes génétiques: le croisement et la mutation. Cependant, étant donné que la représentation des chromosomes est différente, les opérateurs qui les manipulent opèreront aussi de façon différente.

4.2.4.1 Le croisement de regroupement (*Grouping crossover*)

En rappel, le croisement classique tel que défini originellement J.H. Holland, opère selon le mode Couper/Concaténer; dépendamment du domaine d'application, cette façon de procéder peut présenter des inconvénients. En effet, l'application du croisement classique peut entraîner la création d'individus « mal formés ».

Comme l'illustre la figure 6, ce croisement produit quelques fois des chromosomes qui ne peuvent en aucun cas être des solutions à notre problème. Sur cette figure, C1 et C2 sont des chromosomes parents représentant des solutions potentielles; leur croisement donne deux chromosomes fils C1' et C2'.

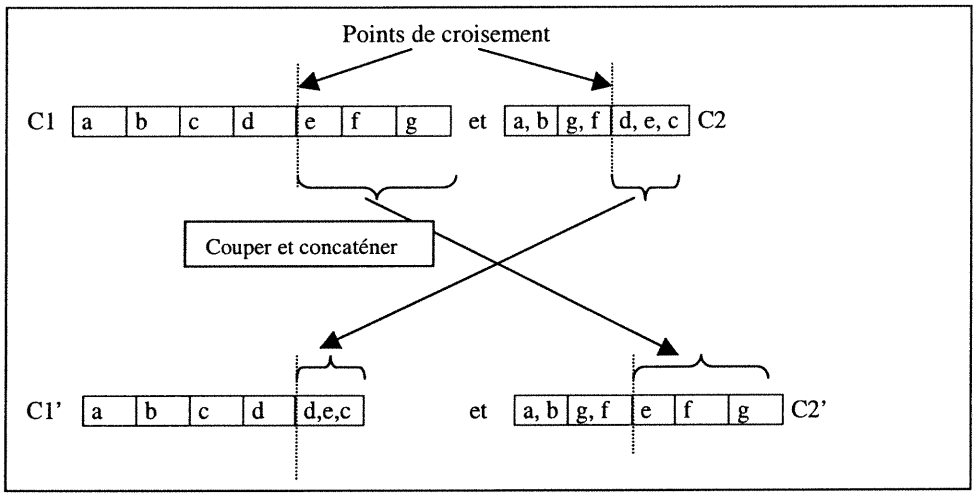


Figure 6. Application du croisement classique.

L'application du croisement classique peut conduire à des solutions qui violent certaines contraintes. Dans ce cas particulier, on constate sur les fils (résultats) C1' et C2' les faits suivants :

- pour le 1er fils : $\cup g_i = \{ a, b, c, d, e\} \neq D$: il y a une violation de la contrainte de complétude; de plus, la donnée *d* se retrouve dans 2 gènes: il y a aussi une violation de la contrainte de cohérence.

- pour le $2^{ème}$ fils $\cup g_i = \{ a, b, g, f, e\} \neq D$ et la donnée *f* se retrouve dans 2 gènes : il y a aussi une violation des contraintes de complétude et de cohérence.

À la lumière de ce qui précède, il est donc nécessaire de définir un croisement qui prenne en compte les contraintes que les groupes formés doivent toujours respecter: les contraintes de cohérence et de complétude.

Pour résoudre ces problèmes, nous allons utiliser le croisement de regroupement tel que défini par E. Falkenauer. À la différence du croisement classique qui opère en Couper/Concaténer comme présenté ci-dessus, le croisement de regroupement opère en trois étapes : Copie/ Injection/ Suppression.

1. *Copie*: les gènes situés à la droite du point de croisement du $2^{ème}$ chromosome, sont recopiés
2. *Injection*: les gènes copiés à l'étape 1 sont injectés dans le 1^{er} chromosome à partir de son point de croisement
3. *Suppression*: les doublons éventuels qui apparaîtront dans le chromosome injecté seront supprimés.

La figure 7 donne une illustration du mode d'opération du croisement de regroupement qui assure le respect des contraintes de cohérence et de complétude.

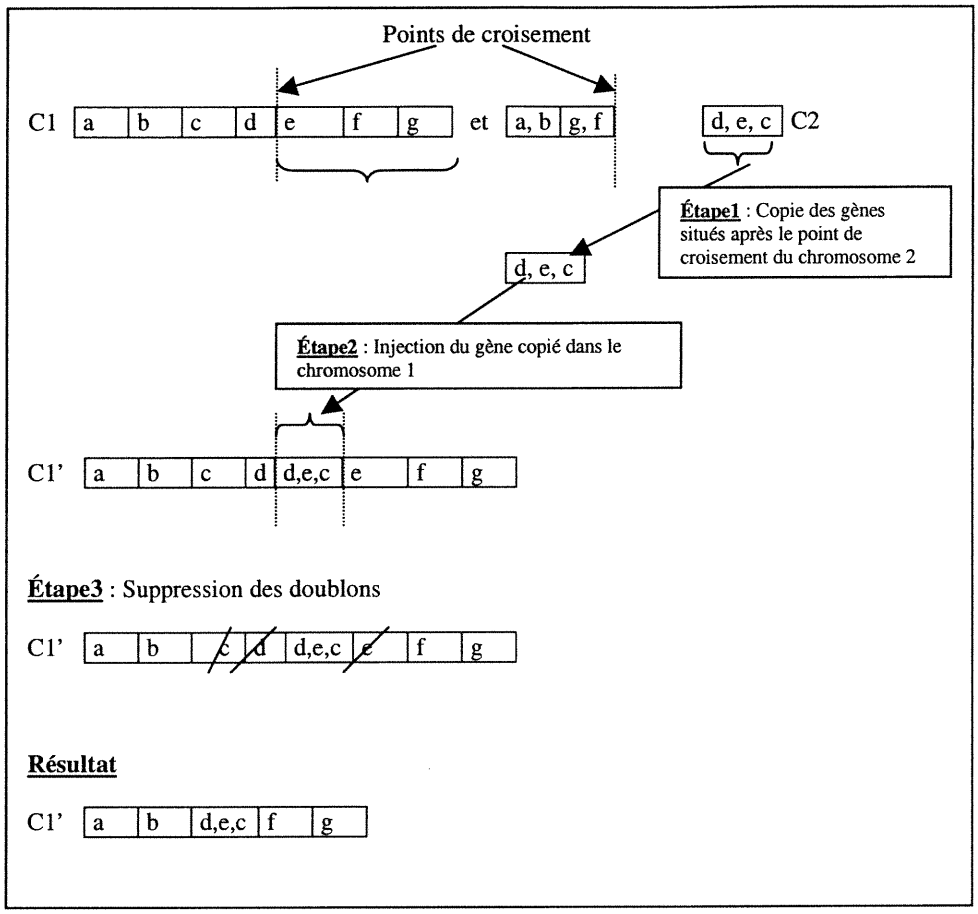
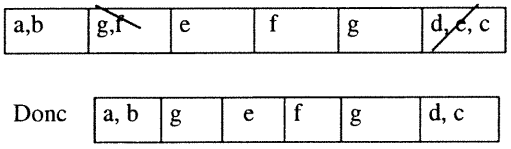


Figure 7. Mode opératoire du croisement de regroupement

On constate que la cohérence et la complétude sont respectées; en effet, aucun gène ne contient deux fois la même donnée et $\cup g_i = \{ a, b, c, d, e, f, g\} = D$.

De même, par application du même principe on obtient le deuxième chromosome fils C2'.

Le 2ème chromosome fils est :



4.2.4.2 La mutation de regroupement

Afin de pouvoir respecter les mêmes contraintes inhérentes aux problèmes de regroupement, il est nécessaire de redéfinir l'opérateur de mutation classique. L'opérateur de mutation de regroupement que nous définissons, diffère largement des mutations classiques notamment des mutations binaires où il suffit de changer la valeur du gène à muter avec une autre valeur possible. En effet, procéder de la même façon que la mutation classique produirait des individus non conformes aux contraintes imposées par les problèmes de regroupement.

La figure 8 illustre ce problème et montre le mécanisme de fonctionnement de la mutation de regroupement.

Illustration :

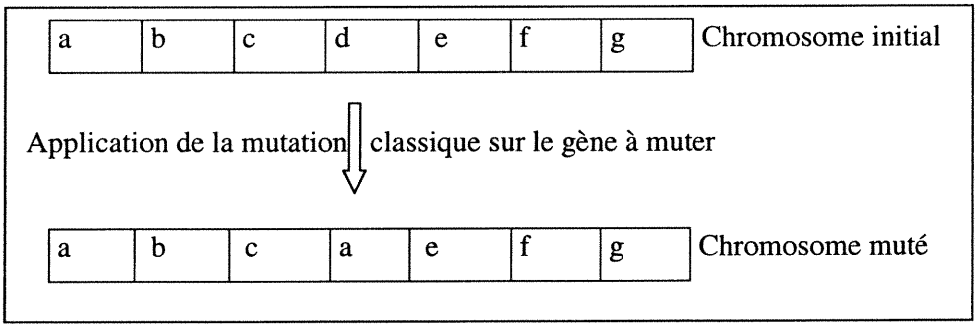


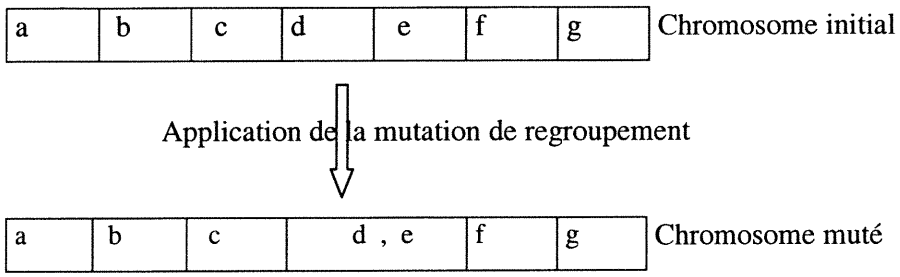
Figure 8. Application de la mutation classique

Nous pouvons constater une violation de la complétude et de la cohérence. En effet:

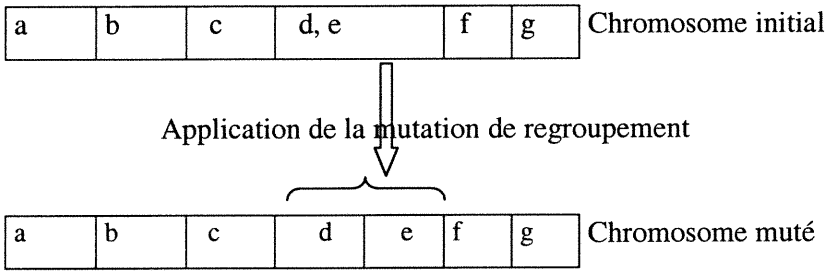
$\cup g_i = \{ a, b, c, e, f, g\} \neq D$ et $\cap g_i \neq \varnothing$ (la donnée *a* est présente dans 2 groupes)

Pour résoudre cela, nous proposons une mutation de regroupement; Elle peut se faire selon deux méthodes.

1- Méthode1 : Fusion de plusieurs groupes en un seul



2- Méthode2 : Scission d'un groupe en deux



Il est important d'observer qu'à la différence de la mutation classique qui opère sur un seul gène, la mutation de regroupement agit sur au moins 2 gènes.

Au sujet de la spécialisation des opérateurs génétiques que nous venons de présenter, nous devons mentionner qu'elle a souvent été préconisée pour résoudre certains types de problèmes; par exemple, Grefenstette a utilisé dans [21] un opérateur de croisement spécialement adapté au problème de commis-voyageur.

4.2.5 Les mécanismes de sélection

Traditionnellement, les algorithmes génétiques offrent plusieurs méthodes de sélection; chacune ayant des avantages et des inconvénients.

1. N/2-élitisme : Les chromosomes sont triés selon la valeur de leurs indices de qualité *(fitness)*. Seule la moitié de la population, correspondant aux meilleurs composants, est sélectionnée. Cette technique a l'avantage d'être facile d'implémentation ; cependant étant donnée que la pression de sélection est très forte, elle peut provoquer une convergence prématurée de l'algorithme. En effet, le maintien d'une diversité génétique suffisante dans la population peut être utile car la combinaison de certains chromosomes de faible qualité peut produire des descendants de très bonne qualité.

2. Méthode de roulette de casino : c'est la plus couramment employée; chaque chromosome occupe un secteur de la roulette dont l'angle est proportionnel à son indice de qualité. Ainsi, un chromosome considéré comme bon, aura un indice de qualité élevé, ainsi qu'un large secteur de la roulette, et par conséquent aura plus de chance d'être sélectionné.

Pour nos expérimentations, nous avons testé les deux méthodes mais les meilleurs résultats ont été obtenus avec la méthode de la roulette de casino.

4.2.6 La fonction d'évaluation des solutions

Rappelons brièvement les faits suivants: dans notre modélisation, les groupes représentent les objets candidats. Notre objectif est de produire des individus de meilleure qualité par sélection et application successives des opérateurs de croisement et de mutation. En outre, dans le cadre des algorithmes génétiques, la qualité de ces individus est évaluée, à l'aide d'une fonction d'évaluation.

Ainsi, pour chaque chromosome C de la population initiale, on applique une fonction d'évaluation qui calcule l'aptitude (appelée *Fitness)* du chromosome. Elle est calculée à partir de la force de chaque gène (groupe) g_i constituant le chromosome.

$$Fitness\ (C) = \frac{\sum_{i=1}^{N} force\ (gi)}{N} \qquad N \neq 0 \qquad (3)$$

Le fitness ainsi calculé, représente la force moyenne d'un chromosome possédant N gènes.

Étant donné que notre objectif est d'obtenir des objets «bien conçus» c'est-à-dire caractérisés par une cohésion élevée et un couplage bas, alors nous pouvons déduire que, la fonction objectif (*force*(g_i)) sera dépendante de deux variables: le couplage et la cohésion. Nous nous retrouvons alors, face à un problème typique de recherche multi-objectifs. En effet, en recherche multi-objectifs, on cherche à optimiser une fonction selon plusieurs critères; ces derniers pouvant être antagonistes. Particulièrement dans le cadre d'une conception, le fait de vouloir minimiser le couplage et maximiser la cohésion peut paraître quelques fois comme des décisions antagonistes.

La situation suivante illustre bien le problème de la définition de notre fonction objectif. Supposons qu'on ait deux groupes g_1 et g_2 possédant les valeurs de métriques suivantes :

* la cohésion de $g_1 = 1$ et le couplage de $g_1 = 0{,}7$; notons la $v(g_1) = (1\ ;\ 0{,}7)$
* la cohésion de $g_2 = 0{,}3$ et le couplage de $g_2 = 0$; notons la $v(g_2) = (0{,}3\ ;\ 0)$

Quelle est la meilleure solution entre $v(g_1)$ et $v(g_2)$? En d'autres termes, comment va-t-on décider qu'un groupe est meilleur qu'un autre, sachant qu'on considère qu'un groupe est meilleur lorsque sa cohésion est élevée et son couplage bas? Il nous faut alors trouver un compromis entre ces critères.

La détermination de la fonction objectif peut être faite de plusieurs façons; ainsi, nous pouvons soit définir une fonction unique qui combine les deux facteurs, soit définir différentes fonctions selon la valeur d'un seuil de décision défini sur un des critères.

a. La fonction objectif vue comme une fonction unique

Le but est de trouver une fonction à 2 variables qui combine les facteurs de décision. Dans la fonction, les critères à optimiser sont combinés de façon ad hoc pour former la fonction objectif. Ce type de fonction prend généralement la forme d'une combinaison (par exemple somme pondérée) des attributs.

Ainsi, si on note F(Cohesion, Couplage) la fonction objectif, elle aura la forme suivante:

$$F(\text{Cohesion, Couplage}) = \alpha\text{Couplage} \perp \beta\text{Cohesion}$$

où α et β représentent des pondérations et $\perp$ l'un des opérateurs arithmétiques classiques (+, -, / ou *)

Le problème avec ce type de fonction est la difficulté à trouver d'une part, la « bonne » combinaison entre ces 2 variables et d'autre part l'opérateur. Cette limitation rend cette technique difficilement applicable dans la réalité. En effet, dans un premier temps, nous avons appliqué cette technique dans nos expériences; la fonction (4) ci-dessous, représente celle que nous avions utilisée. Nous nous sommes retrouvés for limités par l'utilisation de cette fonction dans la mesure où elle ne permettait pas de prendre réellement en compte l'importance relative de chaque facteur.

$$\text{force}(g_i) = (1 + \text{cohesion}(g_i) - \text{couplage}(g_i)) / 2 \qquad (4)$$

b. La fonction objectif vue comme une fonction avec seuil de décision

L'idée principale de cette méthode est de ramener un problème à critères multiples à un problème plus simple avec un seul critère. Ainsi donc, on prendra la décision selon la valeur d'un critère et cela dépendamment d'un seuil défini par rapport à l'autre critère. À partir de ce moment, l'évaluation des groupes devient simple. En effet, en choisissant par exemple comme premier critère la cohésion et en fixant le seuil à MIN_COH, alors en présence de 2 groupes g_1 et g_2, telles que :

$v(g_1) = (cop_1, coh_1)$

$v(g_2) = (cop_2, coh_2)$ où cop_i= valeur de couplage du groupe g_i et coh_i sa valeur de cohésion.

Le choix de la meilleure solution sera fonction des 3 situations suivantes :

1. Si coh_1 et coh_2 >= MIN_COH et $cop_1 > cop_2$ alors g_2 est meilleure que g_1

2. Si coh_1 >= MIN_COH et coh_2 <= MIN_COH alors g_1 est meilleure que g_2

3. Si coh_1 et coh_2 <= MIN_COH et $cop_1 > cop_2$ alors g_2 est meilleure que g_1

Pour respecter ces situations, nous avons besoin de n'importe quelle fonction objectif constituée de 2 fonctions, nommées par exemple f et h, telles que :

- f et h sont des fonctions décroissantes du couplage qui s'appliquent respectivement selon que le facteur cohésion est plus grand ou égal, ou plus petit qu'un seuil fixé;
- f est supérieure à h.

La fonction objectif suivante est un exemple de ce type de fonction:

$$force(gi) = \begin{cases} f(Couplage(gi)) = \dfrac{1}{Couplage(gi)+1} + 1/2 & si\ Cohesion(gi) >= 0.6 \\ h(Couplage(gi)) = \dfrac{1}{4(Couplage(gi)+1)} & si\ Cohesion(gi) < 0.6 \end{cases}$$

Il est important de souligner que dans les deux méthodes d'évaluation de la force des groupes, nous utilisons les mêmes métriques de cohésion et de couplage. Ces métriques sont calculées telles que décrit ci-dessous.

Soient:

$V = \{v_1, v_2, \ldots v_n\}$ un ensemble de n variables;

$R = \{ r_1, r_2, \ldots r_m \}$ un ensemble de m routines accédant aux variables de V;

Les éléments de V et R sont regroupés dans un ensemble G de k groupes, représenté par $G = \{ g_1, g_2, \ldots, g_k \}$;

De plus, $\forall\ V_i$ (un ensemble de variables) $\in g_i$ et $\forall\ V_j$ (un autre ensemble de variables) $\in g_j$, on a $V_i \cap V_j = \varnothing$.

Calcul de la cohésion d'un groupe g_i

La cohésion est calculée à partir de la formule de la métrique Coh[2]. Cette dernière est une variante de LCOM5[40].

$$\text{Cohésion}\quad (\text{gi}) = \frac{\sum_{vj \in gi}^{n} \mu(v_j)}{m_i * n_i}$$

$\mu(vj)$ = nombre de routines qui utilisent la variable v_j

m_i = nombre de routines qui utilisent les variables de V_i

n_i = nombre de variables dans le groupe g_i

Calcul du couplage entre un groupe g_i et les autres groupes (G - g_i)

La formule de calcul du couplage s'inspire de la métrique CBO' de [7] qui est une variante de la métrique CBO (*Coupling Between Object*) ne prenant pas en compte le couplage basé sur l'héritage. Selon CBO, une classe est couplée à une autre, si les méthodes de la classe utilisent des méthodes ou des attributs de l'autre ou vice versa. CBO se trouve donc être définie comme le nombre de classes auxquelles une classe est couplée. Le calcul de couplage revient donc à compter le nombre de liens routines-routines et routines-données entre deux groupes.

Si on note $NRV(g_i, g_j)$ le nombre de fois que les routines dans g_i utilisent les variables dans g_j et $NRR(g_i, g_j)$ le nombre de fois que les routines présentes dans g_i utilisent ou sont utilisées par les routines présentes dans g_j , alors le couplage entre g_i et les autres groupes $(G\text{-}g_j)$ se calcule à partir de la relation suivante:

$$Couplage\ (g_i, G - g_i) = \sum_{j=1,\ j\neq i}^{k} NRV\ (g, g_j) + NRR\ (g_i, g_j))$$

Exemple de calcul de la cohésion et du couplage

Pour illustrer les méthodes de calcul des métriques présentées ci-dessus, appliquons les sur le scénario imaginaire de figure 9;

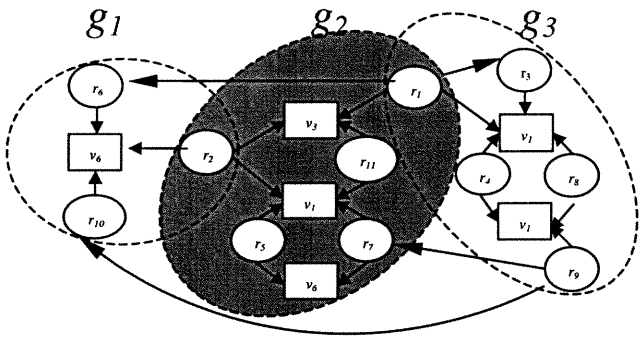


Figure 9. Scénario imaginaire impliquant trois groupes

$$Cohesion\ (g_2) = \frac{\sum_{vj \in g_2} \mu(v_j)}{m_2 * n_2} = \frac{4 + 3 + 2}{5 * 3} = 0.6$$

$$Couplage(g_2, \{g_1, g_3\}) = (NRV\ (g_2, g_1) + NRR\ (g_2, g_1)) + (NRV\ (g_2, g_3) + NRR\ (g_2, g_3))$$
$$= (1 + 1) + (1 + 2) = 5$$

4.2.7 Les critères d'arrêt de l'algorithme

Deux critères d'arrêt sont utilisés : un nombre de générations a été complété ou la convergence est atteinte. Le nombre maximum de générations est choisi suffisamment grand pour observer l'évolution des solutions. La convergence est atteinte dès que l'on constate un taux d'uniformité assez élevée dans la population. En effet, lorsque la population est uniforme, il y a très peu de chance d'avoir une amélioration des solutions dans les générations subséquentes.

4.3 Un exemple d'application de l'algorithme

Pour illustrer notre approche, nous allons l'appliquer sur le code source de *Collections*. *Collections* est un exemple introduit par Canfora[4]. En plus d'être connu dans la littérature, il a l'avantage de ne pas avoir besoin d'autres modules et d'être petit mais suffisamment complexe pour présenter notre approche. D'autres études de cas beaucoup plus complexes seront présentées à la section 5.2 et seront utilisées pour faire la validation de notre approche.

Collections a été écrit en C et manipule une liste, une pile et une file. Comme l'indique la figure 10, le corps de chaque routine a été remplacé par un commentaire indiquant la liste des données utilisées par la routine.

Le tableau III représente la matrice du graphe de références obtenu à partir du programme *Collections*. Comme il apparaît sur le tableau, pour faciliter la lecture des informations qui seront manipulées par l'algorithme nous remplacerons les variables globales par des lettres et les routines par des chiffres.

```
#define MAXDIM 99
typedef int ELEM_T;
typedef int BOOL;
ELEM_T stack_struct[MAXDIM];
int stack_point;
ELEM_T queue_struct[MAXDIM];
Int queue_head, queue_tail, queue_num_elem;
struct list_struct {
        ELEM_T node_content;
        Struct list_struct *next_node;
}list;

main()
{
 /* this program exploits a stack, a queue, and a list of items of type*/
}
/*list of functions with as comment the list of global variables referenced */
void stack_push(el)             {/* stack_point and stack_struct */}
ELEM_T stack_pop()              {/* stack_point and stack_struct */}
ELEM_T stack_top ()             {/* stack_point and stack_struct */}
BOOL stack_Empty()              {/* stack_point */}
BOOL stack_full()               { /*stack_point */}
void queue_insert(el)           {/*queue_struct, queue_head and queue_num_elem*/}
ELEM_T queue_extract()          { /*queue_struct, queue_head and queue_num_elem*/}
BOOL queue_Empty()              { /*queue_num_elem*/}
BOOL queue_full()               { /*queue_num_elem*/}
void list_add(el)               {/* list */}
void list_elim(el)              {/* list */}
BOOL list_is_in()               {/* list */}
BOOL list_empty                 {/* list */}
void global_init                {/* stack_point, list, queue_head, queue_tail and queue_num_elem
                                */}
void stack_to_list()            {/* stack_point, stack_struct and list */}
void stack_to_queue()           {/* stack_point, stack_struct, queue_struct, queue_head and
                                queue_num_elem */}
void queue_to_stack()           {/* stack_struct, queue_tail, queue_num_elem, stack_point and
                                stack_struct*/}
void queue_to_list()            {/* queue_struct, queue_tail, queue_num_elem and list*/}
void list_to_stack()            {list, stack_point and stack_struct}
void list_to_queue()            {/* list, queue_struct, queue_head and queue_num_elem*/}
```

Figure 10. Le programme *Collections*

R'	a stack_struct	b stack_point	c list	d queue_tail	e queue_head	f queue_struct	g queue_ num_elem
1.stack_push	1	1					
2.stack_top	1	1					
3.stack_pop	1	1					
4.stack_empty		1					
5.stack_full		1					
6.stack_to_queue	1	1			1	1	1
7.global_init		1	1	1	1		1
8.list_is_in			1				
9.list_empty			1				
10.stack_to_list	1	1	1				
11.list_to_stack	1	1	1				
12.list_add			1				
13.list_elim			1				
14.queue_to_stack	1	1		1		1	1
15.queue_extract				1		1	1
16.queue_full							1
17.queue_empty							1
18.queue_insert					1	1	1
19.list_to_queue			1		1	1	1
20.queue_to_list			1	1		1	1

Tableau III. Représentation matricielle du graphe de références du programme *Collections*

Un script Discover-Access est exécuté sur le code; le script procède à une analyse syntaxique et fournit en résultat (dans un fichier), une représentation abstraite de ce code. Même si cela n'apparaît pas sur cet exemple, il est important de mentionner que le script fait également un travail d'épuration de données et de routines. Ainsi la « routine *main* » disparaît et il en est de même des routines qui ne manipulent aucune variable. Des détails seront donnés sur ce script dans le chapitre portant sur l'implantation (section 5.1.1).

La population initiale est générée après la lecture du fichier.

La figure 11 montre des chromosomes représentant cette population initiale.

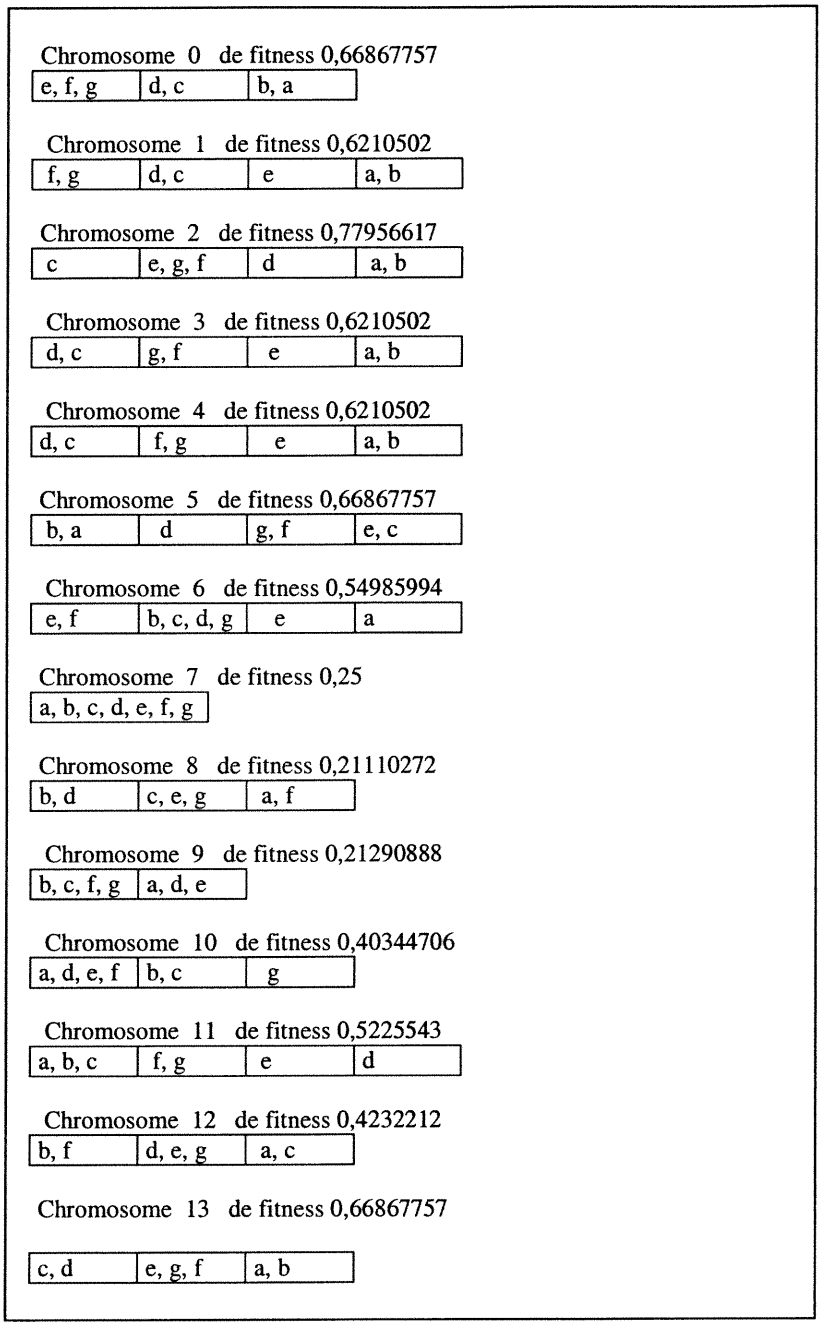


Figure 11. Représentation chromosomique de la population initiale

Il est important de rappeler que, ces représentations chromosomiques apparemment simples renvoient à des réalités plus complexes. Par exemple, le chromosome 0

e, f, g	d, c	b, a

renvoie à la situation suivante :

Le chromosome 0 a pour fitness 0,66867757 et ses gènes sont composés des groupes suivants:

- Groupe 0: de cohésion 0,7037037 et de degré de couplage 0,13157895 contient les données e, f, g
- Groupe 1: de cohésion 0,59090906 et de degré de couplage 0,2 contient les données d, c
- Groupe 2: de cohésion 0,85 et de degré de couplage 0,09411765 contient les données b, a

Nous allons utiliser les paramètres indiqués dans le tableau IV suivant pour effectuer l'exécution de l'algorithme:

Seuil de cohésion	0,6
Probabilité de croisement	0,75
Probabilité de mutation	0,05
Nombre de générations	100
Nombre d'exécutions	10
Taille de la population initiale	2 fois le nombre de variables
Population initiale générée de façon complètement aléatoire?	Non
Méthode de sélection	Roulette de casino

Tableau IV. Paramètres d'exécution

Processus de sélection :

La somme des fitness est $\Sigma Fi = 8,1942$ et la plus grande valeur = 0,77956617.

Pour chaque chromosome Ci, nous calculons la probabilité de sélection pi et la probabilité cumulée q_i telle que $q_i = p1+p2+\ldots+pi$

Chromosomes	Probabilités	Somme Cumulée(qi)
C0	0,0912	0,0912
C1	0,0847	0,1759
C2	0,1063	0,2822
C3	0,0847	0,3669

C4	0,0847	0,4516
C5	0,0912	0,5428
C6	0,075	0,6178
C7	0,0341	0,6519
C8	0,0288	0,6807
C9	0,029	0,7097
C10	0,055	0,7647
C11	0,0713	0,8360
C12	0,0577	0,8937
C13	0,1063	1,0000

Chaque chromosome occupera un secteur de la roulette; l'angle de chaque secteur est proportionnel à la probabilité du chromosome d'être sélectionné, donc à son indice de qualité.

La taille de la population étant 14, on fait tourner 14 fois la roulette pour générer des nombres aléatoires r compris dans l'intervalle [0,1]; si $r < q_0$ alors on sélectionne C0, sinon si $q_{j-1} < r < q_j$ on sélectionne Cj avec $1 \leq j \leq 13$:

r0 :	0,7264	C9 < 0,7264 < C10	10 est sélectionné
r1 :	0,3319	C2 < 0,3319 < C3	3 est sélectionné
r2 :	0,212	C0 < 0,212 < C1	1 est sélectionné
r3 :	0,02788	C2 <0,02788 < C3	3 est sélectionné
r4 :	0,7279	C9 < 0,7279 < C10	10 est sélectionné
r5 :	0,9441	C12 < 0,9441 < C13	13 est sélectionné
r6 :	0,9624	C12 < 0,9624 < C13	13 est sélectionné
r7 :	0,1332	C0 < 0,1332 < C1	1 est sélectionné
r8 :	0,5911	C5 < 0,5911 < C6	6 est sélectionné
r9 :	0,2734	C1 < 0,2734 < C2	2 est sélectionné
r10 :	0,9468	C12 < 0,9468 < C13	13 est sélectionné
r11 :	0,7192	C9 < 0,7192 < C10	10 est sélectionné
r12 :	0,8462	C11 < 0,8462 < C12	12 est sélectionné
r13 :	0,5428	C4 < 0,5428 < C5	5 est sélectionné

Croisement :

On sélectionne 7 (7 = taille de la population / 2) paires de chromosomes (les chromosomes parents) et pour chaque paire, on génère un nombre aléatoire r dans l'intervalle [0,1]; si $r < 0,75$ (0,75=la probabilité de croisement) alors on réalise le

croisement et on insère les chromosomes fils ainsi crées dans la population suivante sinon les parents sont insérés.

On obtient les nombres aléatoires suivants :

0,7641 ; 0,224; 0,8001; 0,953; 0,3454; 0,8265; 0,763; on constate donc que les paires de chromosomes suivantes vont subir un croisement : 1, 3, 4, 6, 7.

La figure suivante résume ce processus de sélection et de croisement; le point de croisement d'un chromosome = taille du chromosome /2.

Paires de chromosomes sélectionnés				Chromosomes-résultats temporaires					Chromosomes finaux après croisement				
a, d, e, f	b, c	g		a, d, e, f	b, c	e	a, b	g	d, f	c	e	a, b	g
d, c	g, f	e	a, b	d, c	g, f	g	e	a, b	d, c	f	g	e	a, b
f, g	d, c	e	a, b	f, g	d, c	e	a, b		f, g	d, c	e	a, b	
d, c	g, f	e	a, b	d, c	g, f	e	a, b		d, c	g, f	e	a, b	
a, d, e, f	b, c	g		a, d, e, f	b, c	a, b	g		d, e, f	c	a, b	g	
c, d	e, g, f	a, b		c, d	e, g, f	g	a, b		c, d	e, f	g	a, b	
c, d	e, g, f	a, b		c, d	e, g, f	e	a, b	a, b	c, d	g, f	e	a, b	
f, g	d, c	e	a, b	f, g	d, c	a, b	e	a, b	f, g	d, c	a, b	e	
e, f	b, c, d, g	e	a	e, f	b, c, d, g	e	a		e, f	b, c, d, g	e	a	
c	e, g, f	d	a, b	c	e, g, f	d	a, b		c	e, g, f	d	a, b	
c, d	e, g, f	a, b		c, d	e, g, f	g	a, b		c, d	e, f	g	a, b	
a, d, e, f	b, c	g		a, d, e, f	b, c	a, b	g		d, e, f	c	a, b	g	
b, f	d, e, g	a, c		b, f	d, e, g	g, f	e, c	a, c	b	d	g, f	e, c	a
b, a	d	g, f	e, c	b, a	d	a, c	g, f	e, c	b	d	a, c	g, f	e

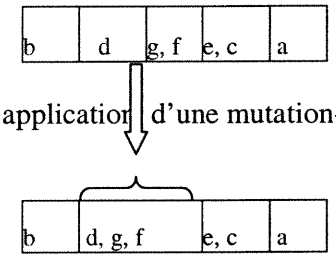
Figure 12. Processus de sélection

Mutation :

Le principe consiste à générer pour chaque gène, un nombre aléatoire *r* compris dans l'intervalle [0,1] et de le comparer au taux de mutation préalablement choisi. Si le nombre *r* est supérieur au taux alors le gène est muté. Pour notre exemple, rappelons que le taux a été défini à 0,05.

Par application de ce principe aux chromosomes de la figure 12, nous obtenons pour le $2^{ème}$ gène du chromosome 13, $r = 0,78 > 0,05$ donc le gène 2 sera muté.

Illustration de la mutation.



Par application d'une mutation-fusion nous obtenons le chromosome suivant :

Finalement le gène muté est réinjecté dans la population avec les chromosomes obtenus à l'étape précédente. La population se trouve donc être composée des chromosomes suivants (voir figure 13):

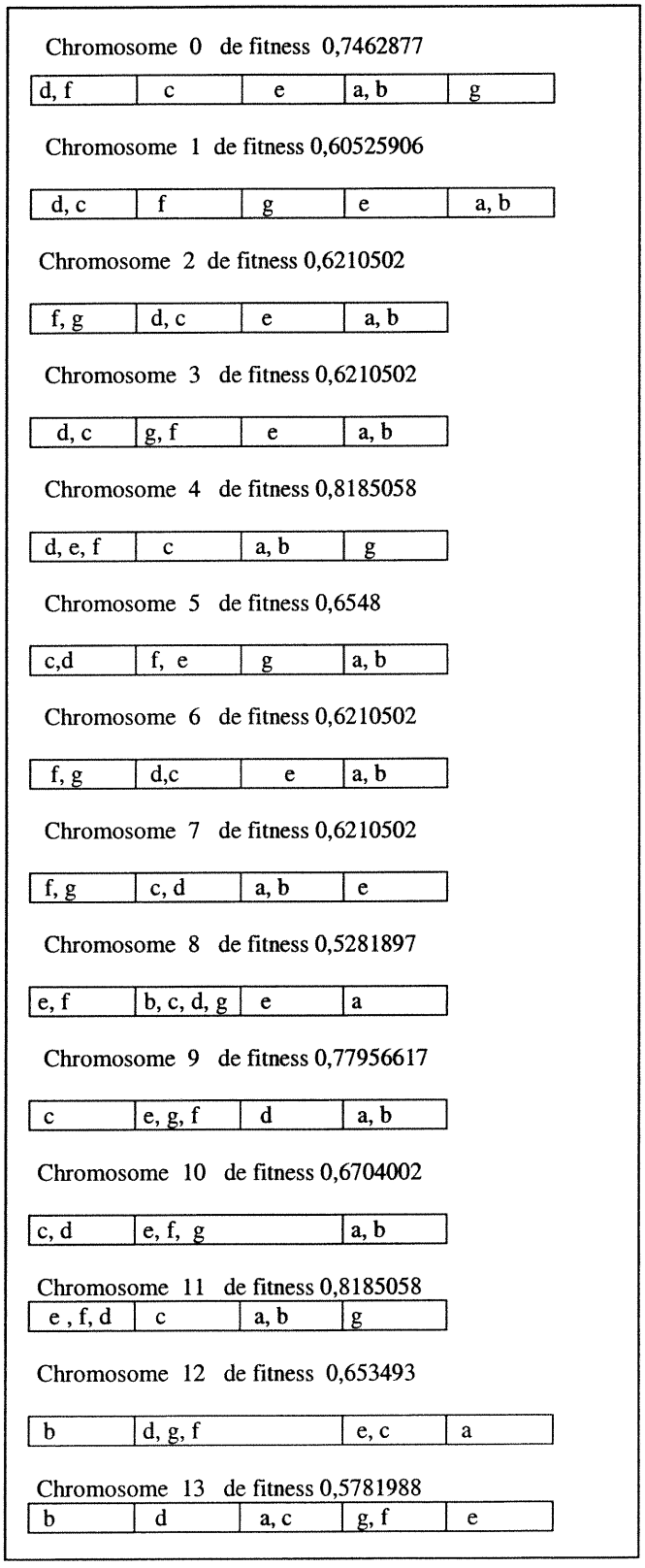


Figure 13. Représentation des chromosomes en cours d'exécution

Une itération de la boucle « TantQue » vient ainsi de se terminer; la figure 13 indique une amélioration de la qualité de la population. En effet, la plus grande valeur du fitness est = 0,8185058 et la somme cumulée = 9,3374.

Après plusieurs exécutions successives, on obtient les résultats finaux qui sont dans la figure 14:

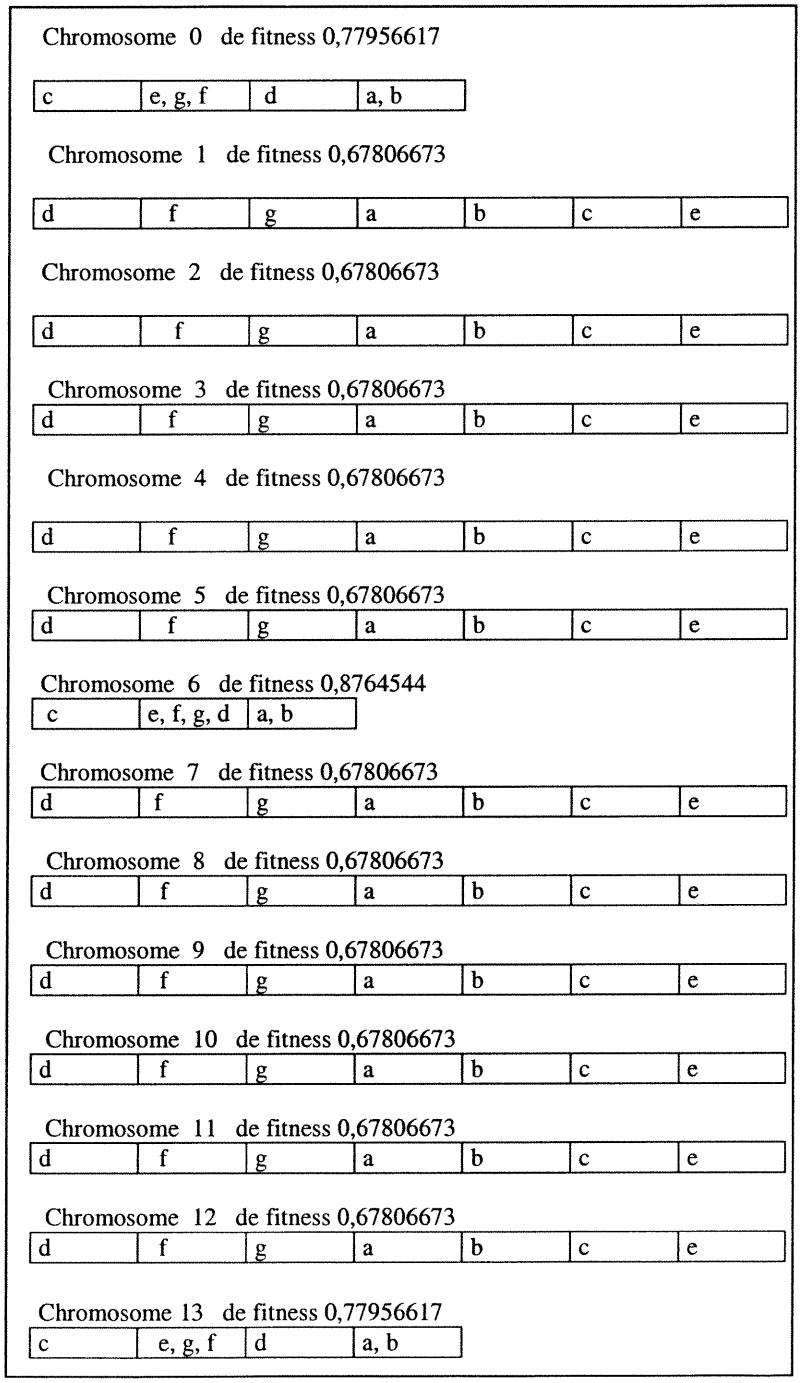


Figure 14. Résultats finaux

La somme cumulée des fitness = 9,8943 et le meilleur chromosome possède une fitness de 0,8764544. Il représente la solution composée des objets-candidats suivants :

O1 = {c} = {list}

O2 = {e, f, g, d} = {queue_head, queue_struct, queue_num_elem}

O3 = {a, b} = {stack_struct, stack_point}

Après l'exécution de l'algorithme, la qualité des objet-candidats proposés peut faire l'objet de validation par un expert, afin de s'assurer de la cohérence et de la vraisemblance des objets proposés. En outre, on remarquera que dans les résultats, les objets-candidats ne font pas référence aux routines du programme. En effet la découverte des méthodes ou des associations entre objets sera l'objet d'étapes ultérieures.

En résumé, l'algorithme que nous avons présenté s'appuie sur des principes simples et éprouvés. En effet, il tire sa simplicité et sa robustesse des algorithmes génétiques en général et des algorithmes génétiques de regroupement de Falkenauer, en particulier. En outre, pour évaluer les objets identifiés, il utilise des propriétés désirables de conception, notamment des métriques de cohésion et de couplage.

Une comparaison avec les autres techniques d'identification d'objets et/ou de migration de codes, place notre algorithme dans le groupe des approches semi-automatiques pour la migration des programmes procéduraux. En effet, son utilisation pour l'identification des objets nécessite de définir un certain nombre de paramètres et de faire valider les résultats par un expert du domaine.

Dans le chapitre qui suit, des détails seront donnés sur le prototype qui implante l'algorithme et sur l'applicabilité de l'approche par rapport à des systèmes réels.

Chapitre 5

Implémentation et validation

Ce chapitre se divise en deux grandes parties; dans la première partie, nous présenterons l'outil qui implante l'algorithme décrit au chapitre précédent. Dans la deuxième partie, nous allons exposer l'approche de validation adoptée puis nous analyserons les résultats obtenus.

5.1 Implémentation

Dans cette section, nous présenterons le prototype qui implante notre approche pour l'identification des objets. Cet outil a été conçu pour être le plus flexible et portable possible. Pour commencer, nous présenterons les outils utilisés pour réaliser le prototype, les choix faits au cours de l'implantation et les interfaces du prototype. Nous présenterons aussi l'outil dont nous nous sommes servis pour faire l'analyse de code.

5.1.1 Les outils utilisés

Le prototype a été implanté en Java notamment en utilisant l'environnement de développement *JBuilder* de Borland. Le choix de Java a été guidé par ses qualités désormais reconnues malgré quelques inconvénients qu'il possède. En effet,

même si Java présente des limitations, notamment en matière de temps d'exécution, il offre par contre la portabilité du code comme le dit si bien son slogan « *Write Once, Run Everywhere* ».

JBuilder a été utilisé parce qu'il offre des outils pour réaliser rapidement des interfaces graphiques.

Pour faire l'analyse du code, nous nous sommes servis de *Discover* mais aussi de *Source Navigator. Discover* est un puissant outil de réingéniérie qui permet d'analyser des applications écrites en C, en C++ ou en Java. Il s'exécute dans l'environnement *Solaris* et offre entre autres les possibilités suivantes:

a. organiser le code selon les types d'entités (fichiers, variables, structures, classes, types de données abstraits, ...) présents dans un programme. La figure 15 illustre cette fonctionnalité.

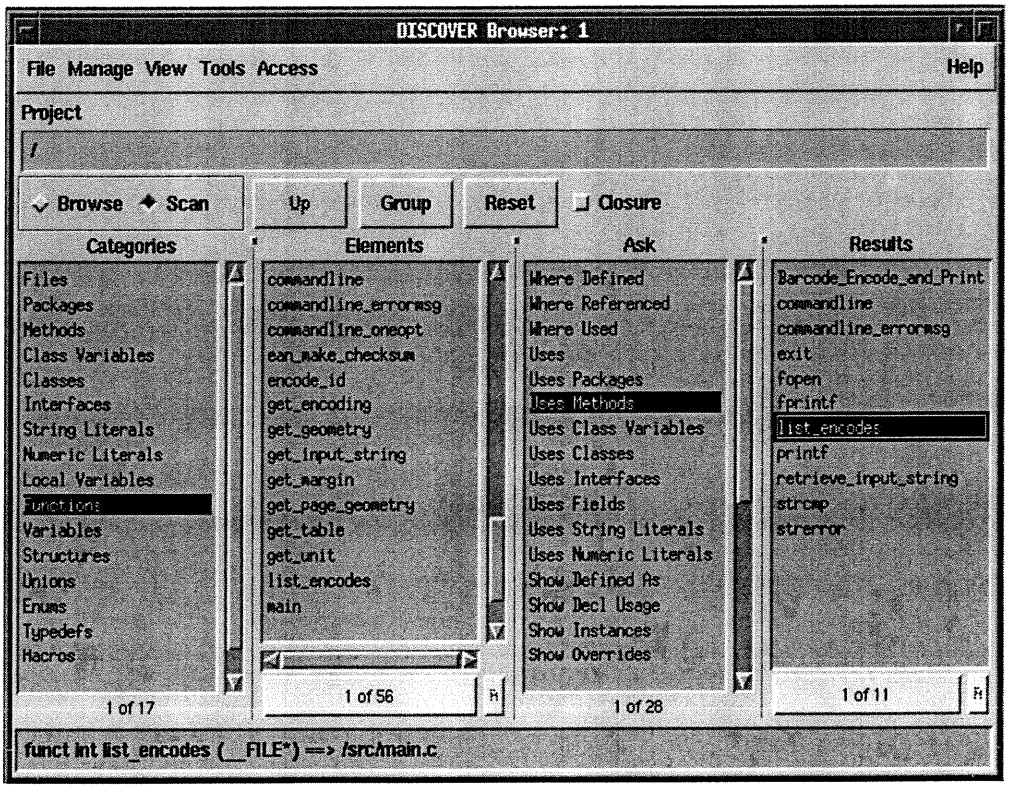


Figure 15. Le *browser* de *Discover*

b. fournir des outils pour visualiser et naviguer à travers le code. Ainsi comme le montre la figure 16, en cours d'analyse, la structure arborescente d'un programme peut être visualisée à tout moment.

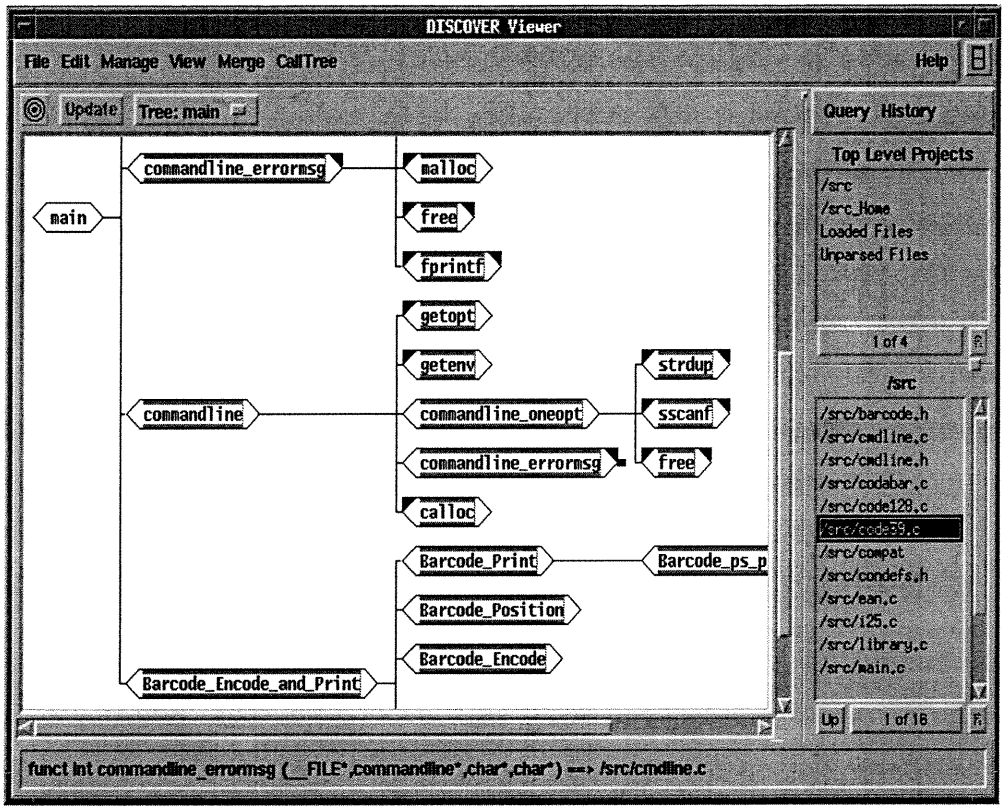


Figure 16. Fenêtre de visualisation de l'arborescence du code

c. fournir des outils pour optimiser le code des applications. En effet, *Discover* permet de calculer différents types de métriques; au nombre de celles-ci on peut citer le nombre de lignes de codes, la complexité cyclomatique, ...

d. faire des requêtes sur les entités présentes dans les programmes sources. Par exemple : lister toutes les super-classes, lister toutes les variables, lister les méthodes satisfaisant à certains critères, ...

Grâce au langage *Discover-Access* inspiré du langage Tcl, *Discover* offre la possibilité d'extraire les informations issues des programmes analysés. Ainsi, à l'aide de ce langage nous avons écrit un utilitaire d'extraction de ces informations. Un extrait du code est fourni à l'annexe 1.
Il faut noter que cet utilitaire ne considère pas toutes les routines et toutes les variables globales. Effectivement, il réalise un travail préliminaire d' «épuration» notamment pour ce qui concerne certaines variables et certaines routines. Ainsi, par exemple les variables globales non utilisées ou les routines n'accédant à aucune variable dans le programme sont ignorées par l'utilitaire. Il en est de même de la « fonction » *Main*. En effet, au cours de nos expériences, nous avons pu constater que la prise en compte du *Main* augmentait de façon non significative le couplage entre les variables. Comme conséquence de cette décision, les variables manipulées uniquement par la fonction *Main* ne seront pas prises en compte.

La non prise en compte de la fonction *Main* se justifie aussi par le fait que du point de vue des opérations relatives au domaine, elle ne réalise pas d'opérations particulières. En effet, le *Main* n'apparaît souvent que dans un rôle de contrôleur.

Cette sélection de données s'inscrit dans le même ordre d'idées que celles préconisées par de nombreux auteurs (voir par exemple [5] et [9]) de séparer les informations relatives au domaine de celles relatives à une implémentation particulière (Cobol, C, ...). Cette étape nécessite très souvent une interaction humaine.

5.1.2 Représentation abstraite des programmes

Les codes sources ne sont pas exploitables directement par l'algorithme; l'utilitaire *Discover-Access* présenté précédemment, permet d'extraire des informations pertinentes contenues dans le code. Il produit dans un fichier texte une représentation plus abstraite du code analysé. Ce fichier contient des informations

exprimant d'une part, des relations du type *«la donnée D_j est utilisée par la routine R_i»*. Ainsi donc, pour chaque donnée on aura la liste des routines qui la manipulent. D'autre part, on retrouve dans le fichier les relations du type « *la routine R_k fait appel à la routine R_l* »

La figure 17 donne une idée plus concrète de la représentation abstraite d'un programme. En rappel, il contient les données D = {a, b, c, d, e, f, g} et les routines R = {r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20}

On peut noter que l'utilitaire insère aussi des commentaires (par exemple le nom du programme analysé) dans le fichier. L'annexe 2 présente le fichier produit pour une des études de cas que nous avons utilisé.

```
@project Canfora_Home
@routinesNames{r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15 r16 r17 r18 r19 r20}

@data-routines relations
a{r1 r2 r3 r6 r10 r11 r14}
b{r1 r2 r3 r4 r5 r6 r7 r10 r11 r14}
c{r7 r8 r9 r10 r11 r12 r13 r19 r20}
d{r7 r14 r15 r20}
e{r6 r7 r18 r19}
f{r6 r14 r15 r18 r19 r20}
g{r6 r7 r14 r15 r16 r17 r18 r19 r20}


@routines-routines relations
r1{r2, r5}
r2{}
r3{r2, r4}
r4{}
r5{}
r6{r4, r16, r18}
r7{r6, r7}
r8{}
r9{}
r10{r4, r12}
r11{r1, r5, r9}
r12{r8}
r13{r8}
r14{r1, r5, r17}
r15{r17}
r16{}
r17{}
r18{r17}
r19{r9, r16, r18}
r20{r12, r17}
```

Figure 17. Représentation abstraite des données et routines du programme *Collections*.

5.1.3 L'interface usager de GOAL

Les principales interfaces sont :

L'écran principal (voir figure 18): il présente les différents menus du prototype. Ainsi, en plus de permettre l'affichage des résultats d'exécution, il offre la possibilité d'activer l'écran de configuration (*Settings*), de sélection du fichier de données, de sauvegarde des résultats d'exécutions, d'aide, ... Une barre d'état permet de savoir à tout moment les paramètres courants d'exécution (probabilité de croisement, de mutation, ...) du programme.

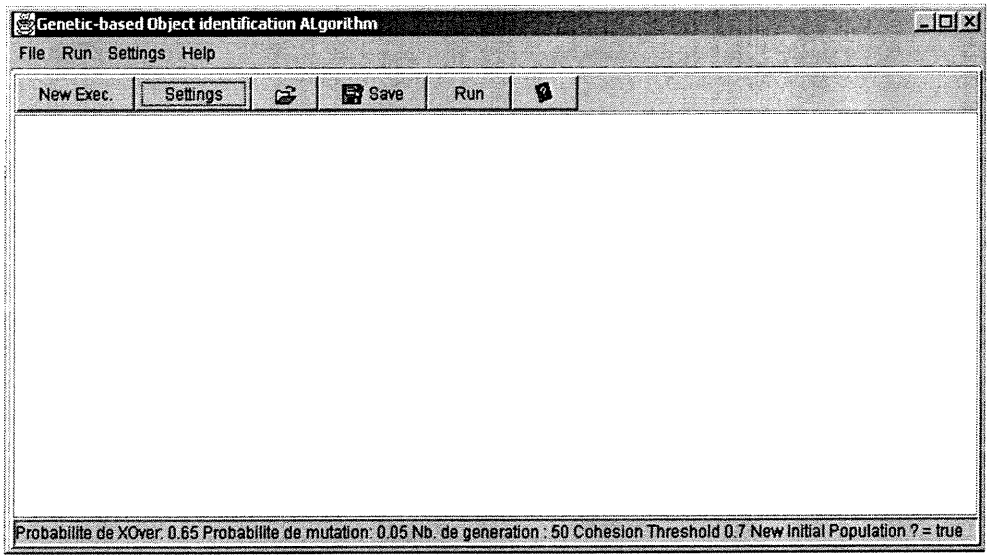


Figure 18. Écran principal de GOAL

L'écran de configuration : comme l'indique la figure 19, il permet de définir les différents paramètres de l'algorithme. Ainsi, il permettra de définir :

a. le nombre de générations souhaité pour l'algorithme
b. la probabilité de croisement
c. la probabilité de mutation
d. le seuil de « bonne cohésion » toléré
e. le choix d'utiliser ou non une population initiale déjà existante.

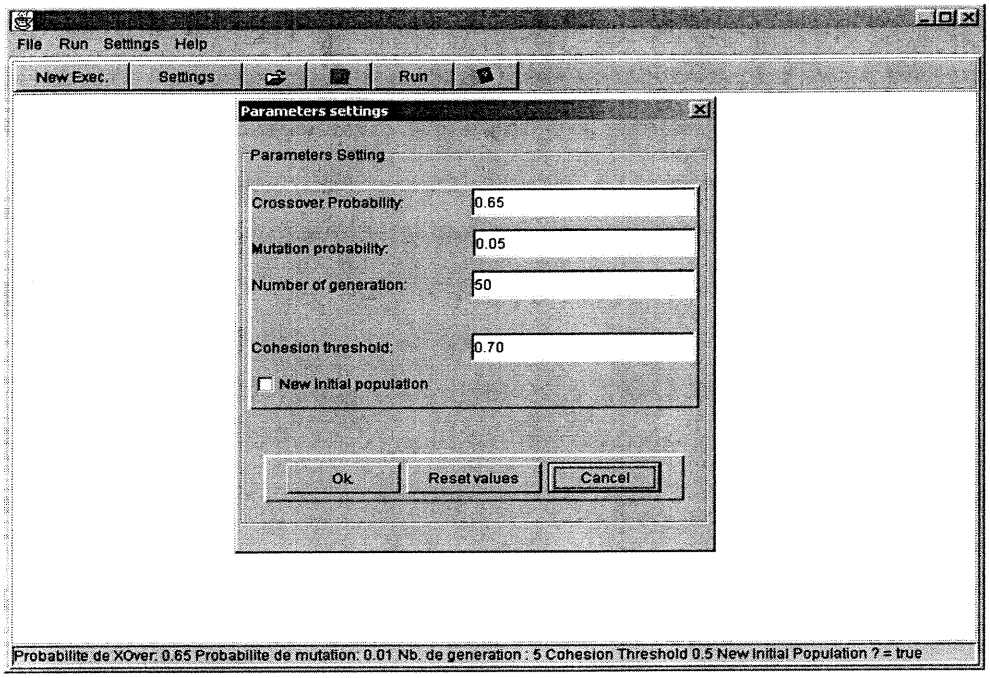


Figure 19. Écran de configuration de GOAL

L'écran d'affichage des résultats d'exécution (voir figure 20): il affiche les résultats intermédiaires de l'exécution et le résultat final de l'algorithme.

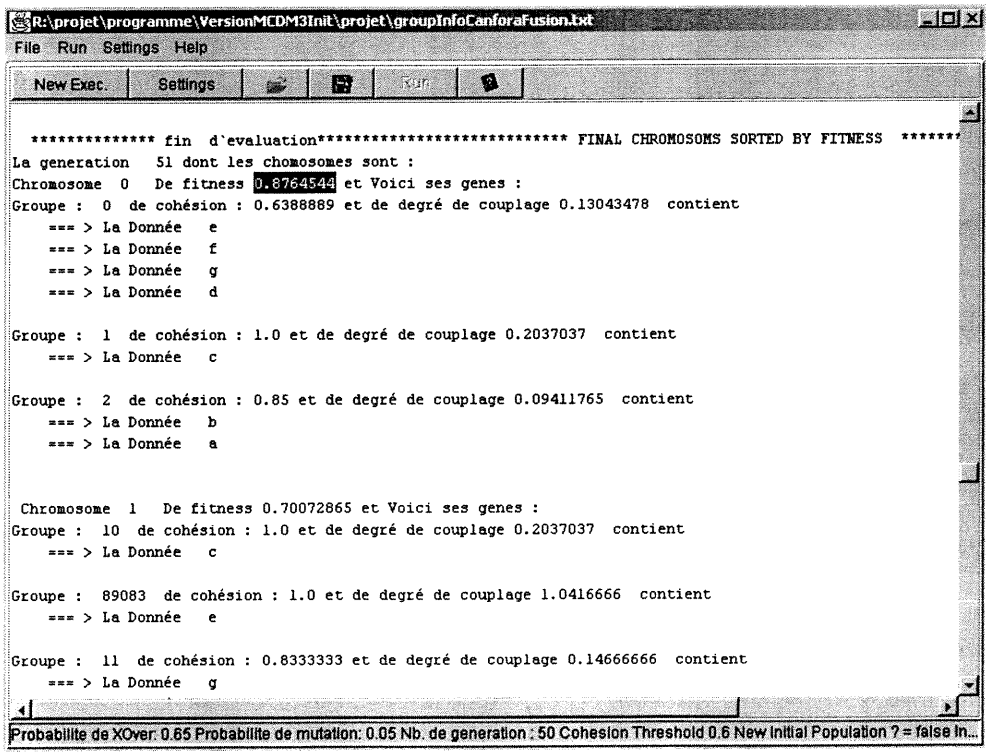


Figure 20. Écran d'affichage des résultats d'exécution

Il faut souligner l'existence du fichier nommé *bestEverFile.txt*. Ce fichier contient le résultat final; en effet, il contient la meilleure solution de toutes les exécutions et un nombre représentant le nombre d'améliorations qui ont eu lieu.

5.2 Validation

Avant de présenter les résultats, nous allons brièvement décrire dans cette section l'approche de validation adoptée. Ainsi, dans un premier temps nous expliquerons les principes de l'expérimentation et la méthode d'évaluation des résultats. Dans un second temps, nous donnerons une description des systèmes utilisés et procéderons à la présentation et à l'analyse des résultats.

5.2.1 Principes de l'expérimentation

L'approche globale de validation que nous avons adoptée s'inspire d'une technique très employée dans les validations basées sur les heuristiques(voir [29]). Cette technique consiste à soumettre différentes entrées au programme à valider et à comparer les résultats obtenus aux résultats attendus.
Pour utiliser cette technique de validation, il est impérieux, comme le signale Jalote dans [26], soit de posséder les résultats attendus (diagrammes de classes, documentation, ...), soit de disposer d'un expert du domaine pour valider les résultats. En outre, il est nécessaire de définir un mécanisme d'évaluation des résultats.

Pour ce qui nous concerne, la validation s'est faite en utilisant surtout la première stratégie. En effet, pour certaines applications nous avions à notre disposition les versions procédurales et orientées objet des systèmes à migrer et pour d'autres nous avons utilisé les documents des processus de rétro ingénierie réalisés par des étudiants gradués. Effectivement, la validation de nos expériences s'est faite en collaboration avec les étudiants du cours gradué de génie logiciel IFT 6251: Sujets

spéciaux en génie logiciel. Dans le cadre de leurs projets, les étudiants ont réalisé manuellement la rétro ingénierie de certains systèmes que nous avons utilisés dans l'étape de validation. Ils se sont servis dans leurs tâches, d'outils d'analyse de code tels que *Source Navigator* et de leurs connaissances des domaines.

Pour l'évaluation des résultats, nous avons établi des critères d'évaluation de la qualité des objets identifiés par notre prototype. Ces critères sont les mêmes que ceux proposés par Girard et son équipe dans [16]. Les objets-candidats (C) et les résultats attendus encore appelés objets-références (R) sont comparés dans le but de déterminer d'une part les objets-candidats qui sont bons, c'est-à-dire qui établissent une correspondance 1 à 1 avec les résultats attendus, et d'autre part ceux qui n'établissent pas de correspondance 1 à 1 avec les résultats attendus mais qui sont suffisamment proches des résultats attendus pour être considérés dans un processus de migration.
Ainsi, un objet-candidat C est considéré comme étant « assez bon » pour être utile dans un processus de migration, s'il partage avec un objet attendu R, au moins 70% des attributs. Cette situation se note $\mathbf{C} \subseteq_p \mathbf{R}$. Dans [28] Koschke affirme que, bien qu'arbitraire le choix de ce pourcentage est motivé par le fait que les 70% signifient que pour qu'un objet-candidat possédant quatre attributs soit acceptable il faut qu'au moins trois de ses attributs soient présents dans un objet-référence.
Partant de ce qui précède, les objets-candidats identifiés ont été évalués et classés en 3 catégories: bon, acceptable et mauvais.

- **Catégorie « bon »** : un objet-candidat C est classé dans cette catégorie, si la correspondance entre ce dernier et un objet-référence R est quasiment parfaite c'est à dire qu'on a : $C \subseteq_p R$ et $R \subseteq_p C$.

Dans un processus de migration, les objets-candidats présents dans cette catégorie ne nécessiteront qu'une vérification rapide dans le but d'identifier au besoin, les attributs qui doivent être rajoutés ou supprimés de ces objets. Ainsi donc, ces objets-candidats ne nécessiteront que des modifications mineures pour correspondre à des objets du domaine.

- **Catégorie «acceptable » :** la relation se fait dans une seule direction c'est à dire qu'on a l'une des situations suivantes:
  - **i.** au moins 70% des attributs de l'objet-candidat C sont présents dans l'objet attendu R alors que moins de 70% des attributs de R le sont dans C, c'est-à-dire qu'on a $C \subseteq_p R$ et pas $R \subseteq_p C$.
  - **ii.** la situation contraire c'est à dire plus de 70% des attributs de R sont dans C et moins de 70% des attributs de C sont dans R; on a donc $R \subseteq_p C$ et pas $C \subseteq_p R$.

Dans un processus de migration, les objets de cette catégorie nécessiteront de la part de l'expert du domaine beaucoup plus d'attention; de manière générale, pour aboutir à des objets du domaine à partir des objets de cette catégorie, ce dernier devra procéder par scission ou fusion de ces objets.

- **Catégorie «mauvais »:** la relation ne se fait dans aucune direction c'est-à-dire que les objets-candidats partagent moins de 70% de leurs attributs avec les objets résultats et vice versa; les objets de ce groupe ne sont pas assez proches des résultats attendus. Ils représentent généralement des objets qui ont été identifiés mais qui n'existent pas. Dans [16], Girard appelle cette catégorie, la catégorie des « faux positifs »[3].

Pour la plupart des objets de cette famille, cela signifie que les attributs ont été mis ensemble par pure coïncidence. Les objets classés dans ce groupe seront très souvent ignorés dans un processus de migration.

Exemple. En considérant l'exemple de la figure 21, C_1 et R_1 forment une «bonne correspondance»; puisque nous avons $C_1 \subseteq_p R_1$ et $R_1 \subseteq_p C_1$, donc C_1 est dans la « catégorie bon ». L'objet-référence R_3 (avec $R_3 \cap R_1 = \varnothing$) est un sous ensemble de C_1. En référence à R_2, C_2 se classe dans la catégorie des « acceptables »; il en

[3] Voir terminologie anglaise de *False Positive*

est de même pour C_3. Par contre, C_4 ne correspond à aucune référence; il fait donc partie de la catégorie des faux positifs .

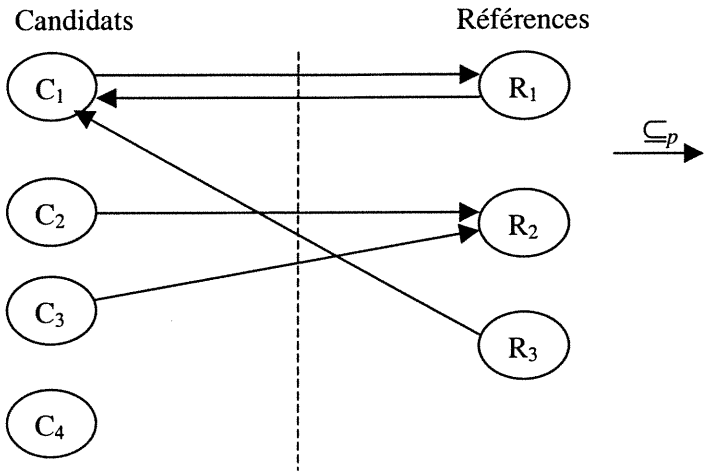


Figure 21. Exemple de correspondances entre objet-candidats et objet-référence.

5.2.2 Description des données de l'expérimentation

D'entrée de jeu, il faut insister sur le fait que nous avons rencontré des difficultés pour obtenir les codes sources de systèmes de grande taille écrits en C. Néanmoins, nous avons pu tester notre solution sur des systèmes de taille moyenne. Ultérieurement, le cas des systèmes de grande taille sera envisagé dans le cadre d'un processus distribué pour l'identification des objets dans les grands systèmes. Du reste, dans [38] nous avons présenté une ébauche de solution. Dans la section consacrée à la présentation des perspectives de ce travail, nous donnerons un aperçu de l'approche globale envisagée.

Pour ce qui concerne ce travail, les systèmes utilisés ont été pour la plupart obtenus à partir de l'Internet notamment sur le site de GNU[24]. Ainsi, le prototype présenté à la section 5.2 a été testé sur trois systèmes de taille moyenne écrits en C.

Le tableau V donne un récapitulatif des caractéristiques de ces différents systèmes. La présente section sera consacrée exclusivement à la présentation de chacun de ces systèmes.

5.2.2.1 Cas 1 : Le système Barcode

Le système Barcode est une bibliothèque écrite en C par Alessandro Rubini[36]. C'est un outil de conversion de chaînes de caractères en barres imprimables. Il supporte de nombreux standards d'encodage de chaînes textuelles (UPC, EAN, ISBN, CODE39, MSI, ...); l'exécution de Barcode produit en résultat un fichier post-script. Comme déjà souligné, ce système a été analysé à la main par les étudiants du cours IFT6251.

5.2.2.2 Cas 2 : Le système Jalote

Le système Jalote est un logiciel de programmation des cours offerts par une université. Le code source est le même que celui utilisé par Pankaj Jalote dans [26]. Ce système a l'avantage de posséder aussi bien une version orientée objet, qu'une documentation assez fournie; en effet, le dossier des exigences logicielles et le document de conception sont fournis avec le code source. Le modèle objet utilisé pour faire la validation a été obtenue à partir du code source orienté objet.

5.2.2.3 Cas 3 : Le système SGA-C

Le système SGA-C [42] est une traduction et une extension de la version originale de l'algorithme génétique présenté par Goldberg en 1989. Il a été écrit par Robert E. Smith et son équipe; Cette version possède des caractéristiques additionnelles mais ses opérations sont essentiellement identiques à la version originale écrite en Pascal par Goldberg. Cependant, une des différences que les auteurs de SGA-C mentionnent dans leur rapport est le mode de représentation des chromosomes. En

effet, SGA-C représente les chromosomes sous forme de chaînes de bits à un niveau machine; ceci a pour conséquence d'accroître l'efficacité des traitements mais malheureusement de rendre la représentation trop liée au type de machine utilisé. Ils proposent à cet effet un module pour redéfinir la représentation adéquate dépendamment de la machine et du compilateur C utilisés.

Systèmes	Lignes de code	Nombre de variables globales	Nombre de routines
Barcode	3800	56	56
Jalote	1718	19	40
SGA-C	1151	28	42

Tableau V. Récapitulatif des principales caractéristiques des systèmes analysés.

5.2.3 Application et résultats de l'expérimentation

Comme nous l'avons mentionné antérieurement, le prototype implantant notre approche a pour fondement les algorithmes génétiques. Ceci a pour conséquence, la nécessité de bien choisir les paramètres classiques des algorithmes génétiques à savoir les taux de mutation et de croisement, la taille de la population, la technique de génération de la population initiale,... Dans notre cas particulier d'identification d'objets à partir des algorithmes génétiques, il faut rajouter un autre paramètre clé: la valeur seuil de cohésion des objets candidats.
La détermination de la taille de la population et de la probabilité de mutation ont fait l'objet de nombreux travaux (voir par exemple [19]) dont les détails ne seront pas donnés dans le présent mémoire.

Pour chaque système, nous avons testé notre prototype en faisant varier les seuils de cohésion. Pour chaque seuil nous avons procédé à une vingtaine d'exécutions afin d'obtenir des résultats statistiquement significatifs. Ainsi, les seuils de cohésion suivants ont été utilisés : 0,6 , 0,5 et 0,4.

Afin de déterminer la solution pour une série d'exécutions utilisant le même ensemble de paramètres (seuil de cohésion, taux de mutation, taux de croisement, ...), à chaque exécution la solution obtenue est comparée avec la solution de l'exécution précédente; lorsque l'on constate une amélioration, l'ancienne solution est remplacée par la nouvelle.

Les résultats obtenus lors de nos expériences sont synthétisés dans les tableaux de la page suivante (voir tableaux VI, VII, VIII). On peut également apprécier de façon visuelle à l'aide des histogrammes (voir figures 22, 23, 24), l'allure de l'évolution des solutions en fonction des paramètres.

a. Résultats sur le système Barcode

Valeurs de cohésion	Résultats (%)		
	Bon	Acceptable	Mauvais
0,6	57,14	28,57	14,29
0,5	37,50	37,50	25,00
0,4	55,56	22,22	22,22

Tableau VI. Résultats obtenus sur le système Barcode

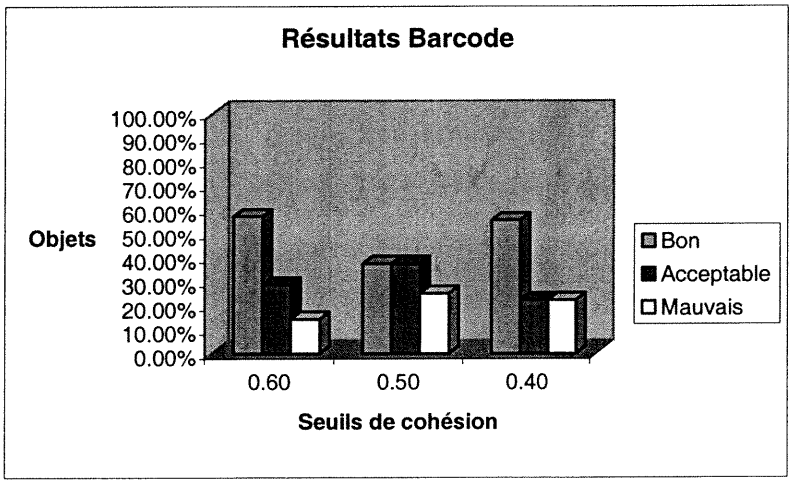


Figure 22. Histogramme des résultats d'exécution sur le système Barcode

b. Résultats sur le système Jalote

Valeurs de cohésion	Résultats (%)		
	Bon	Acceptable	Mauvais
0,6	27,27	36,36	36,36
0,5	22,22	44,44	33,33
0,4	33,33	16,67	50

Tableau VII. Résultats obtenus sur le système Jalote

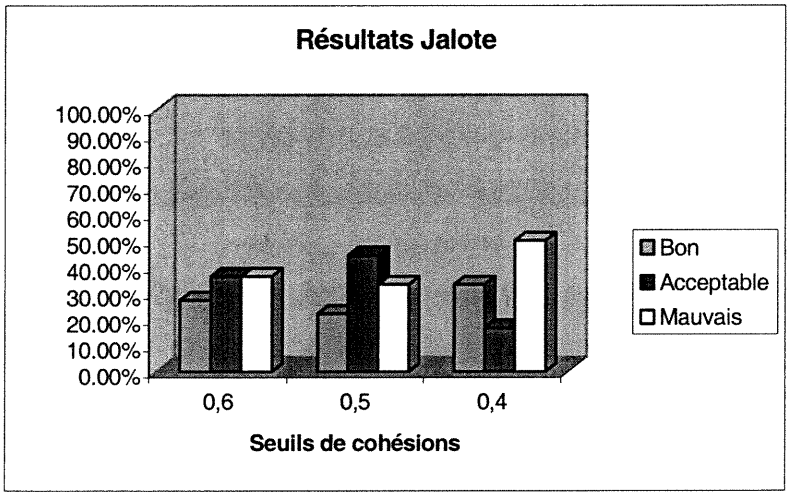


Figure 23. Histogramme des résultats d'exécution sur le système Jalote

c. Résultats sur le système SGA-C

Valeurs de cohésion	Résultats (%)		
	Bon	Acceptable	Mauvais
0,6	4,35	86,95	8,70
0,5	12,5	75	12,5
0,4	0	40	60

Tableau VIII. Résultats obtenus sur le système SGA-C

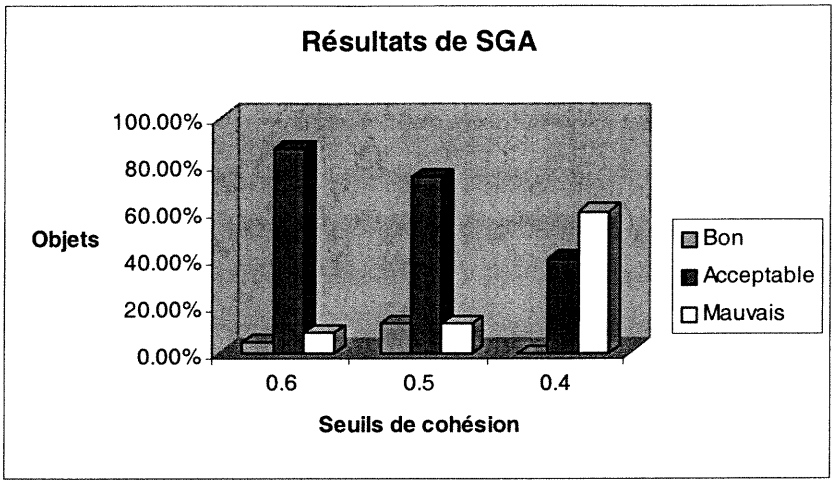


Figure 24. Histogramme des résultats d'exécution sur le système SGA-C

5.2.4 Analyse des résultats

De prime à bord, à partir des résultats qui précèdent on peut faire les constatations suivantes :

1. pour un même système, les résultats varient en fonction du seuil de cohésion.
2. il n'existe pas dans l'absolu une valeur de cohésion qui donne toujours le meilleur résultat quelque soit le système.

Par exemple, dans nos expériences alors que pour le système Barcode c'est une valeur de cohésion égale à 0,6 qui nous a donné les meilleurs résultats, pour les systèmes Jalote et SGA-C c'est le seuil de cohésion 0,5 qui nous a donné les meilleurs résultats.

Cependant, nous avons constaté qu'une cohésion de 0,5 permet d'avoir des résultats satisfaisants pour tous les systèmes.

À partir des constatations qui précèdent, nous pouvons faire deux remarques :

1. Avec des paramètres bien choisis, GOAL arrive à identifier la majorité des objets attendus. Ceci positionne les algorithmes génétiques de regroupement comme une alternative crédible pour l'identification des objets.

2. Pour déterminer les bons paramètres de l'algorithme, dépendamment de chaque système, nous avons dû exécuter plusieurs fois l'algorithme. Cela démontre encore une fois dans un processus de migration d'applications léguées, l'importance de l'étape de caractérisation présentée à la section en 3.1.

Comme nous l'avons mentionné antérieurement, l'étape de caractérisation des systèmes, permet de déterminer des informations capitales sur le système et le contexte de migration. Il ne fait aucun doute, que si cette étape avait été complétée, elle aurait permis par exemple, de mieux définir la valeur de la cohésion ou d'utiliser différentes heuristiques dépendamment des systèmes. En effet, un « bon seuil » de cohésion ne peut être fixé de façon unilatérale. C'est ainsi que même s'il est souhaitable d'avoir une cohésion élevée, l'expérience montre que dans la pratique, il est très rare d'atteindre des valeurs de cohésion de 1 ou même de 0,8. En effet, une valeur de cohésion égale à 1 signifierait que toutes les méthodes accèdent à toutes les variables: ce qui est très rare dans une classe.

Une seconde analyse des résultats fait ressortir d'autres faits tout aussi pertinents. Ainsi, dans le tableau IX, nous synthétisons pour chaque système, les meilleurs résultats. En outre, la colonne «pourcentage des objets identifiés» (catégorie bon + catégorie acceptable) est mise en comparaison avec celle de la catégorie des « faux positifs ».

Systèmes	Pourcentage des objets identifiés (Bons + Acceptables)	Pourcentage des faux positifs
Barcode	85,71	14,29
Jalote	66,67	33,33
SGA-C	91,30	8,70

Tableau IX. Récapitulatif des résultats des tests

Dans les trois cas, plus des 2/3 des objets identifiés sont satisfaisants. Cependant, on constate que les résultats du système Jalote sont relativement bas par rapport aux autres.

Une analyse plus approfondie révèle que les trois systèmes ont des niveaux de qualité différents (voir tableau X). SGA-C apparaît comme étant assez bien codé; en effet, des types de données abstraits y sont utilisés pour limiter l'accès à certaines variables. De plus, très peu de variables de travail (par exemple les *flags*) y sont employées. Enfin, SGA-C possède une bonne modularité: ses modules ont des valeurs de cohésions élevées et les couplages inter-modules sont assez bas.
Le système Barcode aussi possède une bonne modularité et l'accès aux principales données est restreinte. Cependant, de nombreuses variables intermédiaires sont utilisées pour gérer la conversion entre les différents types d'encodages qu'il manipule.
À l'opposé des deux précédents systèmes, Jalote apparaît au contraire comme un système pauvre du point de vue des trois critères. Cette analyse permet de conclure que notre algorithme fonctionne bien quand les systèmes à migrer sont de qualité bonne ou acceptable mais lorsque le système est très mal codé, GOAL ne peut produire des miracles[38].

Systèmes	Modularité	Variables de travail	Encapsulation
SGA-C	+	+	+
Barcode	+	-	+
Jalote	-	-	-

Tableau X. Analyse de la qualité des systèmes

Chapitre 6 Conclusion

6.1 Synthèse

Bien que la littérature en matière d'approches d'identification d'objets dans les systèmes légués soit relativement abondante, force est de reconnaître que la plupart souffrent de certaines limites. En effet, l'identification des objets dans le code procédural est loin d'être une activité évidente. Aussi, la majorité des techniques proposées apparaissent-elles comme des aides pour la migration. Soit elles proposent des objets qui doivent être validés par des experts du domaine, soit elles traitent des situations qui ne sont pas toujours conformes à la réalité. Par exemple, les approches basées sur les graphes recherchent les sous graphes isolés pour en faire des objets. Ces approches ne fonctionnent très bien que pour les cas de programmes idéals déjà conçus selon une approche orientée objet[4].

Dans ce mémoire, nous avons présenté GOAL, un algorithme génétique pour l'identification des objets dans les applications léguées; nous avons aussi décrit le prototype qui l'implante. GOAL se distingue des autres approches essentiellement par le fait qu'il intègre des critères de qualité dans son processus. Bien qu'étant en mesure de fonctionner de façon automatique, il se classe comme une approche semi-automatique. Cette caractéristique le rend particulièrement très efficace.

En effet, lorsqu'un expert du domaine est disponible pour faire un bon choix des paramètres d'exécution, notre technique fournit des résultats de très bonne qualité.
GOAL se base largement sur la théorie des algorithmes génétiques définis par Holland et adapté par Falkenauer pour les problèmes de regroupement desquels fait partie celui de l'identification des objets dans les systèmes légués.
En outre, il est l'illustration de l'utilisation d'une technique éprouvée d'intelligence artificielle à un autre type de problème apparemment très différent. Ainsi, face à un problème désormais classique de génie logiciel, nous avons apporté une solution combinée d'intelligence artificielle et de génie logiciel.

De la théorie des algorithmes génétiques, GOAL hérite sa robustesse. Par ailleurs, comme nous l'avons largement exposé au chapitre 5, le prototype construit peut être utilisé comme le prolongement d'une autre méthode d'identification d'objets. En effet, au lieu d'utiliser les données brutes d'un programme de base, notre prototype peut utiliser en entrée les résultats d'une première application d'une autre approche, notamment celle de Shen[41].

Au cours de ce travail, nous avons également pu mesurer l'importance relative des métriques de couplage et de cohésion dans un objet et par extension dans une classe. De plus, notamment grâce à la fonction d'évaluation de la qualité des objets, nous avons pu combiner ces deux métriques à première vue contradictoires pour obtenir un agrégat significatif.

L'algorithme en l'état ne nécessite pas des modifications majeures. Cependant, le prototype qui l'implante pourrait être amélioré. En effet, pour obtenir de meilleurs résultats il serait judicieux de procéder encore à certains calibrages; ceux-ci porteront notamment sur les critères tels que la taille de population, la politique de gestion de la population, les conditions d'arrêt,... La fonction d'évaluation pourrait aussi être raffinée pour avoir une fonction plus expressive de la qualité des objets.

6.2 Travaux futurs

Nonobstant la qualité des objets identifiés, GOAL offre de nombreuses pistes pour son extension. Au nombre de celles-ci nous avons celle de son utilisation dans les systèmes de grande taille, celle du paramétrage de la valeur seuil d'une bonne cohésion et conséquemment celle de la fonction objectif.

Face à des systèmes de très grande taille, des problèmes de performance peuvent se poser quelques fois. En effet, les temps d'exécution notamment de génération de la population initiale et de croisement des individus peuvent limiter l'applicabilité de notre algorithme. Pour résoudre ces problèmes de performance, on pourrait s'appuyer sur une approche de migration incrémentale utilisant des implantations parallèles efficaces. Du reste, cette stratégie est en cours d'exploration au sein de notre laboratoire[38]. À chaque exécution, une partie du système sera migré et une interface devrait permettre aux modules d'inter communiquer.

Comme la plupart des algorithmes génétiques, GOAL nécessite de posséder certaines connaissances afin de « bien configurer » ses nombreux paramètres (taille de la population, méthodes de sélection, taux de mutation, ...); une amélioration, notamment en ce qui concerne son utilisabilité, serait l'usage des « algorithmes génétiques peu paramétrés » proposés par Harik et Lobo dans [22].

Pour finir, une autre piste à explorer serait celle concernant la détermination des seuils d'une bonne cohésion et d'un bon couplage. Il se dégage des travaux relatifs à la détermination du seuil d'un bon niveau de couplage ou de cohésion que ce seuil est une valeur floue. Ainsi, dans le but d'améliorer la fonction objectif basée sur le seuil de cohésion, on pourrait appliquer les principes de la logique floue. Cela va consister à définir des fonctions d'appartenance qui seront associées à la cohésion et au couplage; les termes linguistiques de cette fonction seront par exemple *Bon, Moyen* et *Mauvais*. Ainsi, l'évaluation du niveau de couplage ou de

cohésion d'un objet sera déterminer à partir de son degré de vérité par rapport aux fonctions d'appartenance.

Bibliographie

[1] Alliot J-M. thèse d'habilitation INPT. Techniques d'optimisation stochastique appliquées à certains problèmes du trafic aérien, 1996, ENAC, France.

[2] Briand L., Daly J. and Wüst J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering Journal*, 1998, pp. 65-117.

[3] Barbut M., Monjardet B., *Ordre et classification. Algèbre et Combinatoire, tome II*, Hachette, 1970.

[4] Canfora G., Cimitile A. and Visaggio G. Assessing modularization and code scavenging techniques, *Journal of Software Maintenance: Research and Practice*, 1995, pp. 317-331.

[5] Canfora G., Cimitile A. and Munro M. An improved algorithm for identifying objects in code, *Software Practice and Experience*, 1996, pp. 25-48.

[6] Canfora G., Czeranski J. and Koschke R., Revisiting the Delta IC Approach to Component Recovery. Proceedings of the Working Conference on Reverse Engineering, 2000, pp. 140-149.

[7] Chidamber S. R., Kemerer C. F. Towards a metrics suite for Object Oriented Design. Proceedings of Conference on Object-oriented Programming, Systems, Languages and Applications (OOPSLA'91), 1991, pp. 197-211.

[8] Davey B. A. , Priestley H. A. Introduction to Lattices and Order, Cambridge University Press, 1992.

[9] De Lucia A., Cimitile A., Di Lucca G. A. and Fasolino A. R. Identifying Objects in Legacy Systems Using Design Metrics, *Journal of Systems and Software*, 1999, pp. 199-211.

[10] De Lucia A., Giuseppe A., Di Lucca, Fasolino A. R., Guerra P. Petruzzelli S. Migrating Legacy Systems towards Object-Oriented Platforms. Proceedings of International Conference on Software Maintenance, 1997, 122-129.

[11] Dunn F. M. and Knight J. C. Automating the Detection of Reusable Parts in Existing Software. Proceedings of International Conference on Software Engineering, 1993, pp. 381-390.

[12] Falkenauer E. *Genetic algorithms and grouping problems*. John Wiley and Sons, 1998.

[13] Fergen H., Reichelt P. and Schmidt K. P. Bringing objects into COBOL: MOORE - a tool for migration from COBOL85 to object-oriented COBOL. Proceedings of Conference on Technology of Object-Oriented Languages and Systems, 1994, pp. 435-448.

[14] Fleurent C. et Ferland J. Algorithmes génétiques hybrides pour l'optimisation combinatoire, 1996, RAIRO pp. 373-398.

[15] Gall, H.C. and Klösch, R. R. Finding objects in procedural programs. Proceedings of Working Conference on Reverse Engineering, 1995, pp. 208-217.

[16] Girard J-F., Koschke R. and Schied G. A Metric-based Approach to Detect Abstract Data Types and State Encapsulations. Proceedings of Automated Software Engineering Conference, 1997, pp. 82-89.

[17] Godin R. and Mili H., Building and Maintaining Analysis-Level Class Hierarchies using Galois Lattices. Proceedings of Conference on Object-oriented Programming, Systems, Languages and Applications, 1993, pp. 394-410.

[18] Godin R., Mineau G., Missaoui R., Mili H. Méthodes de classification conceptuelle basées sur les treillis de Galois et applications, Revue d'Intelligence Artificielle, 1995, pp. 105-137.

[19] Goldberg D. E. and Deb K. A Comparative analysis of selection schemes used in genetic algorithms. Foundations of Genetic Algorithms, 1991, pp. 69-93.

[20] Goldberg D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading MA Addison Wesley, 1989.

[21] Grefenstette J. J. Incorporating Problem Specific Knowledge into Genetic Algorithms cité dans [14].

[22] Harik G. R. and Lobo F. G. A parameters-less genetic algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), 1999, pp. 258-265.

[23] Holland J. H. *Adaptation in Natural and Artificial Systems* , The University of Michigan Press, 1975.

[24] http://www.gnu.org

[25] Jacobson I. and Lindstrom F. Re-engineering of old systems to an object-oriented architecture. Proceedings of Conference on Object-oriented Programming, Systems, Languages and Applications, 1991, pp. 340-350.

[26] Jalote P. *An Integrated Approach to Software Engineering.* Springer Verlag 1996.

[27] Kaufman L. and Rousseeuw P. J. *Finding Groups in Data: An introduction to Cluster Analysis.* John Wiley, 1990.

[28] Koschke R. Ph.D. Thesis, Atomic Architecture Component Recovery for Program Understanding and Evolution, Institute for Computer Science, University of Stuttgart, 2000.

[29] Lakhotia A. and Gravley J. M. Towards Experimental Evaluation of Subsystem Classification Recovery Techniques. In Proceedings of the Working Conference on Reverse Engineering, 1995, pp. 262-269.

[30] Lehman M. M. and Belady L. A. *Program Evolution*, New York, Academic Press, 1985.

[31] Liu S.S. and Wilde N. Identifying objects in a conventional procedural language : An example of data design recovery. Proceedings of International Conference in Software Maintenance, 1990, pp. 226-271.

[32] Missaoui R., Godin R. and Sahraoui H. A. Migrating to an Object-Oriented Database Using Semantic Clustering and Metamodelling techniques, In *Data and Knowledge Engineering Journal*, 1998, pp. 97-113.

[33] Newcomb P. and Kottik G. Reengineering procedural into object-oriented systems. Proceedings of Working Conference on Reverse Engineering, 1995, pp. 237-249.

[34] Panas E. Livadas and Johnson T. A new approach To findings Objects in Programs, Software Maintenance: *Research and Practice*, 1994, pp. 249-260.

[35] Pressman R. 1987. *Software Engineering : A Practionner's Approach*, 2nd edition. McGraw-Hill.

[36] Rubini A. A Library for drawing bar codes, 2000. http://www.gnu.org/software/barcode/barcode.html

[37] Sahraoui H., Lounis H., Melo W. and Mili H. A Concept Formation Based Approach to Object Identification in Procedural Code, Automated Software Engineering 1999, pp. 387-410.

[38] Sahraoui H., Konkobo I., Shen S., Wu Lei and Valtchev P., From Legacy to OO Code: A Quality-Driven Migration Approach, 2001.

[39] Schwanke R. W. An Intelligent Tool for re-engineering software modularity. Proceedings of International Conference on Software Engineering, 1991, pp. 83-92.

[40] SEMA Group, *Fast Programmer's Manual*, France, 1997.

[40b] Sedgewick, R. *Algorithms*, Addison Wesley, Reading, MA.

[41] Shen S., Master thesis, Object Identification Using Conceptual Clustering, Université de Montréal, 2001.

[42] Smith R. E., Goldberg D. E. and Earickson J. A., A C-Language Implementation of a Simple Genetic Algorithm, TCGA Report No. 91002, 1991.

[43] Siff M. and Reps T. Identifying modules via concept analysis. Proceedings of

International Conference on Software Maintenance, 1997, pp. 170-179.

[44] Snelting G. Concept analysis: A new framework for program understanding. In Proceedings of the Workshop on Program Analysis for Software Tools and Engineering (PASTE'98), 1998, pp. 1-10.

[45] Snelting G. and Tip F. Reengineering class hierarchies using concepts analysis. In Foundations of Software Engineering, 1998, pp. 99-110.

[46] van Deursen A. and Kuipers T. Identifying Objects using Cluster and Concept Analysis. Proceedings of IEEE Automated Software Engineering Conference, 1999, pp. 98-106.

[47] Wille R. Concept Lattices and Conceptual Knowledge Systems. Computers Mth. Appli., 1992, pp. 493-515.

[48] Zimmer J. A. Restructuring for style, *Software Practice and Experience*, 1990, pp. 365-389.

Annexe 1 : Script Access-Discover d'extraction d'informations dans les systèmes à migrer

```
set fileId [open /u/konkoboi/projet/scriptDiscover/outputfile w
0600]
puts $fileId "@project [cname [home_proj]]"
set fun_size [size [defines -funct /]]
set pfun [sort [defines -funct /]]

############################################################

#construction des informations sur la liste des routine(liste des
noms de routines)
set i 0
set fonctionMain main

puts -nonewline $fileId "@routinesNames\{"
foreach fun $pfun {
  set funstring [cname $fun]
  if { [string compare $funstring $nmain] !=0 } {
    if { $i == 0 } {
      puts -nonewline $fileId "[cname $fun]"
    } else {
      puts -nonewline $fileId " [cname $fun]"
    }
    incr i 1
  }
}
puts $fileId "\}\n"


############################################################

# Constitution de la matrice des données ( pour chaque variable
# on liste les routines qui le manipulent )

puts $fileId "@dataInfo"
set data_set [sort [defines -variables /]]

foreach data $data_set {
   puts -nonewline $fileId "[cname $data]\{"
   set datafun [sort [filter funct [where used $data]]]
   set argfuns [sort [filter funct [get_instance_statement
[instances $data]]]]
   set ufuns [sort [set_union $argfuns $datafun]]

   set j 0
   foreach sfun $ufuns {
     set singlefunstring [cname $sfun]
     if { [size [where defined $sfun]] != 0 && [string compare
$singlefunstring $nmain] != 0 } {
       if { $j == 0} {
         puts -nonewline $fileId  "[cname $sfun]"
       } else {
         puts -nonewline $fileId  " [cname $sfun]"
       }
```

```
        incr j 1
      }
    }
    puts $fileId "\}"
}
puts $fileId "\n"


#################################################################

# Constitution de la matrice des routines ( pour chaque routine
# on liste les routines avec lesquelles il a un lien )

puts $fileId "\@routinesInfo"

set k 1
foreach fun $pfun {
  set funstring [cname $fun]
  if { [string compare $funstring $nmain] !=0 } {
    puts -nonewline $fileId "[cname $fun]\{"
    set psubfun [sort [ uses -functions $fun ]]
    set psubfun_size [ size $psubfun ]
    set n 0
    foreach subfun $psubfun {
      set subfunstring [cname $subfun]
      if { [size [where defined $subfun]] != 0 && [string compare
$subfunstring $nmain]!= 0 } {
        if { $n == 0 } {
          puts -nonewline $fileId "[cname $subfun]"
        } else {
          puts -nonewline $fileId " [cname $subfun]"
        }
        incr n 1
      }
    }
    if { $k == $fun_size } {
      puts -nonewline $fileId "\}"
    } else {
      puts $fileId "\}"
    }
  }
  incr k 1
}
close $fileId
```

Annexe 2: Fichier de données produit par le code Access-Discover et utilisé en entrée du prototype : cas du système Jalote.

```
@@ project jalotc_Home
@@ routinesNames{PgNoPrefSchLable chk_cap chk_dup_course_no
chk_dup_room chk_dup_sched_course chk_dup_time_slot
 chk_fmt_course_no chk_fmt_room_no chk_fmt_time_slot
display_courses display_pref display_rooms display_scheds
 display_times form_pref_list get_course_index get_next_line
get_room get_room_index get_slot_index
 get_validated_input input_error is_safe_allotment print_TimeTable
print_conflicts print_explanation
 print_output sched_pg_no_pref sched_pg_pref sched_ug_no_pref
sched_ug_pref schedule separate_courses
 sort_rooms validate_class_rooms validate_dept_courses
validate_file1 validate_file2 validate_lec_times
}

@@ dataInfo
ClassroomDB{display_rooms get_room get_validated_input
print_TimeTable print_explanation print_output
sched_pg_no_pref sched_pg_pref sched_ug_no_pref sched_ug_pref
schedule}
ConflLst{print_output schedule}
CourseDB{display_courses display_scheds get_course_index
get_validated_input print_TimeTable
print_conflicts print_explanation print_output schedule
validate_file2}
ExplnLst{print_output schedule}
PgAlloc{print_conflicts print_explanation sched_pg_no_pref
sched_pg_pref sched_ug_pref schedule}
PgNP{display_scheds get_room is_safe_allotment sched_pg_no_pref
sched_ug_pref separate_courses}
PgP{sched_pg_pref}
SchCourses{display_pref display_scheds get_validated_input
sched_pg_no_pref schedule}
TimeTable{print_output schedule}
TimeslotDB{display_pref display_times form_pref_list
get_validated_input print_TimeTable print_conflicts
print_explanation print_output schedule validate_file2}
TotConfl{print_conflicts sched_pg_no_pref sched_pg_pref
sched_ug_no_pref sched_ug_pref schedule}
TotCourses{display_courses get_course_index validate_dept_courses
validate_file2}
TotExpln{print_explanation sched_pg_pref sched_ug_pref}
TotRooms{PgNoPrefSchLable display_rooms get_room is_safe_allotment
print_TimeTable sched_pg_no_pref sched_pg_pref
sched_ug_no_pref sched_ug_pref validate_class_rooms}
TotSchCourses{chk_dup_sched_course display_pref display_scheds
form_pref_list get_course_index get_next_line
get_room sched_ug_no_pref separate_courses validate_file2}
TotTimes{PgNoPrefSchLable display_times form_pref_list
get_slot_index print_TimeTable sched_pg_no_pref
sched_ug_no_pref validate_lec_times}
```

```
UgNP{display_scheds get_room sched_pg_no_pref sched_ug_no_pref
sched_ug_pref separate_courses}
UgP{display_scheds get_room sched_pg_pref sched_ug_pref
separate_courses}
line_no{chk_cap chk_fmt_course_no chk_fmt_room_no
chk_fmt_time_slot form_pref_list get_next_line input_error
validate_class_rooms validate_dept_courses validate_file2
validate_lec_times}


@@ routinesInfo
PgNoPrefSchLable{sched_pg_no_pref}
chk_cap{}
chk_dup_course_no{}
chk_dup_room{}
chk_dup_sched_course{}
chk_dup_time_slot{}
chk_fmt_course_no{}
display_courses{}
display_pref{}
display_rooms{}
display_scheds{display_pref}
display_times{}
form_pref_list{get_slot_index}
get_course_index{}
get_next_line{}
get_room{}
get_room_index{}
get_slot_index{}
get_validated_input{separate_courses validate_file1
validate_file2}
input_error{}
is_safe_allotment{PgNoPrefSchLable}
print_TimeTable{}
print_conflicts{}
print_explanation{}
print_output{print_TimeTable print_conflicts print_explanation}
sched_pg_no_pref{get_room}
sched_pg_pref{get_room}
sched_ug_no_pref{get_room}
sched_ug_pref{PgNoPrefSchLable get_room is_safe_allotment}
schedule{sched_pg_no_pref sched_pg_pref sched_ug_no_pref
sched_ug_pref}
separate_courses{}
sort_rooms{}
validate_class_rooms{chk_cap chk_dup_room chk_fmt_room_no
get_next_line input_error sort_rooms}
validate_dept_courses{chk_dup_course_no chk_fmt_course_no
get_next_line input_error}
validate_file1{validate_class_rooms validate_dept_courses
validate_lec_times}
validate_file2{chk_dup_sched_course form_pref_list
get_course_index get_next_line}
validate_lec_times{chk_dup_time_slot chk_fmt_time_slot
get_next_line input_error}
```