Thèse de Doctorat en
Informatique

Algorithmes Génétiques Interactifs pour le Text-Retrieval

Yann LANDRIN-SCHWEITZER

Rapporteurs	Michèle Sebag	LRI
	Gilles Venturini	Université de Tours
Directeur de Thèse	Evelyne Lutton	INRIA
Examinateurs	Pierre Collet	Université du Littoral, Calais
	Pierre Parisot	Novartis Pharma
	Thérèse Vachon	Novartis Pharma

Table des matières

Index

Préface

Lorsque j'ai commencé cette thèse, en 1999, l'objectif était de répondre à un problème industriel très précis qui, par l'ampleur des données à traiter et des moyens pour le faire, ne semblait avoir que des relations très distantes avec les moteurs de recherche grand public. Mais en trois ans, les technologies et les usages ont beaucoup évolué, et les moteurs de recherche ont pris un rôle de premier plan dans l'accessibilité aux réseaux d'information. En parallèle, la personnalisation et l'adaptabilité des outils logiciels sont entrés dans les préoccupations des éditeurs. C'est pourquoi il m'est aujourd'hui beaucoup plus aisé de montrer l'intérêt de cette recherche.

Il faut donc souligner la clairvoyance des chercheurs du groupe IK@N de Novartis d'avoir eu l'idée de cette étude, début 1999, quand l'importance même des outils de recherche dans l'exploitation des informations était encore floue. Plus encore, je les remercie de m'avoir fait confiance, dans les méthodes utilisées pour – tenter – d'explorer ce domaine. Thérèse Vachon, Pierre Parisot, Nicolas Grandjean et Thomas Welker m'ont été d'une grande aide, tant pour comprendre les techniques du text-retrieval et de l'analyse textuelle, que sur le plan matériel.

Un grand merci aussi à ma directrice de thèse, Evelyne Lutton, qui m'a supporté (au sens propre comme au figuré) durant ces trois années. Je lui dois d'avoir eu des conditions de travail aussi confortables, et surtout beaucoup de souplesse dans l'organisation de mes travaux.

Merci aussi à tous ceux dans le laboratoire Fractales qui m'ont apporté un peu plus que de simples relations professionnelles : chaleur, intérêt, amitié ...

Merci enfin aux membres de mon jury, en particulier aux rapporteurs, Michèle Sebag et Gilles Venturini, qui par leurs remarques m'ont permis de clarifier et formaliser les résultats de ma recherche, étape indispensable pour être en mesure d'aller au-delà et de construire sur cette base.

Chapitre 1

INTRODUCTION

Résumé du Chapitre.

La croissance en volume, l'enrichissement et la spécialisation des bases de données documentaires ont modifié les exigeances d'accès aux informations. L'utilisateur ne demande plus simplement d'obtenir toutes les données sur un sujet – en éliminant les redondances – mais un ensemble de données pertinentes au sein de son contexte personnel.

Les liens qu'il est possible d'établir entre les informations[1] prennent une grande importance dans l'évaluation de leur pertinence. Ce qui complique encore la tâche d'exploration et d'analyse des ensembles documentaires.

Les outils de text-retrieval[2] tentent de répondre à ces besoins d'une part en enrichissant les résultats de requêtes d'outils de navigation et de visualisation[3], et d'autre part en développant les outils de profilage des utilisateurs.

Beaucoup de techniques de profilage reposent sur des modèles des utilisateurs (qu'ils soient cognitifs ou statistiques, voir section 2.1). Nous tenterons de montrer dans cette thèse qu'il est possible de fournir un contenu et une présentation personnalisés des résultats de recherche sans modèle comportemental explicite. Nous abordons ce problème sous l'angle d'un apprentissage de profils, formulé comme une optimisation de procédures de recherche. La pauvreté

[1] Au sens général : liens hypertexte, liens sémantiques, origine commune (auteur, organisme, éditeur...), similitude de structure, etc.

[2] Nous utiliserons ce terme dans l'ensemble du document pour désigner la recherche de documents textuels, le terme "recherche textuelle" pouvant être utilisé dans d'autre contextes, et par là prêter à confusion.

[3] Par exemple, présentation en grappes des ensembles documentaires, navigation par proximité sémantique... voir [VacPar+01] pour un aperçu.

des informations disponibles et la complexité de l'espace de recherche nous ont conduit à étudier l'emploi d'Algorithmes Evolutionnaires pour cette tâche.

1 Web, intra/inter-net et bases de données électroniques

1.1 Volumes de données électroniques, spécialisation, cross-linking

A l'origine réseau de la défense américaine (ARPAnet, en 1969), puis étendu aux universités et centres de recherche (UCLA, UCSB, University of Utah, et SRI, dès 1970), le réseau internet n'était dans l'esprit de ses concepteurs qu'un système de transport de données. Mais la dernière décennie du vingtième siècle a vu son extension à l'échelle planétaire, portée par la diffusion rapide d'informations et les activités commerciales qu'il permet.

Parallèlement ont été introduits des formats de "marquage" de documents ou de données de plus en plus élaborés. Le langage SGML [SGML], en 1986, dont l'objectif était essentiellement de faciliter la mise en valeur de sections appropriées de documents texte, a ouvert la voie à HTML [HTML-2.0]. Langage de mise en forme graphique, il permet aussi de **relier** des documents entre eux.

La généralisation du support matériel (le réseau internet) et d'un mode de présentation des documents permettant d'en suivre la structure (le réseau de liens entre documents HTML) a fait du "web" l'aspect le plus visible du réseau mondial à partir de 1990. Devenu une source incontournable d'information, son mode de consultation et sa structure s'imposent peu à peu aux autres supports de données électroniques. Depuis 1995, les réseaux d'entreprises se présentent de plus en plus sous forme d'"intranets", version localisée de l'internet. Ils sont devenus à la fois le vecteur des échanges d'information interpersonnels, et le support des ressources documentaires.

Au succès du web s'est adjoint la généralisation des bases de données relationnelles (RDB), qui prend son origine dans les travaux de Codd, en 1970 [Cod70]. A l'origine issues d'un besoin d'archivage, de mise en relation et de catégorisation des données, pour permettre leur traitement automatique, elles sont apparues comme l'instrument idéal permettant de générer de manière automatique des documents publiables sur internet ou un intranet. L'apparition de PHP [@PHP] en 1999 a généralisé ce modèle.

Les deux modèles se sont rencontrés dans une version **très** étendue du format HTML : le langage XML [XML-1.0] (standardisé par les spécifications du W3C [@W3C]), permettant à la fois de relier et catégoriser des ensembles de données. Cette mise en relation peut être effectuée à une échelle très fine (celle du mot pour les documents textuels) et selon différents critères (sémantique, temporelle, géographique, etc...).

La convergence de ces technologies, la généralisation des accès internet – au moins au sein des entreprises– et la multiplicité des services fournis grâce à ce média ont eu deux conséquences principales. La première est l'inflation du volume d'informations disponible, tant via internet qu'au sein des intranets. La seconde est l'intégration dans les habitudes de travail du mode de parcours –de "navigation"– utilisé sur internet, en suivant des liens contextuels.

Internet, un océan d'information...

Une évaluation récente (2001) estimait qu'internet donnait accès à plus de 2 milliards de documents. Une grande partie de ces données provient de sites d'entreprise, qui utilisent internet comme un vecteur peu coûteux de communication et de diffusion de leurs catalogues. Internet étend même les possibilités de communication, en regroupant dans un même support les possiblilités des différents médias traditionnels (presse, télévision) et en leur ajoutant des aspects interactifs.

La conversion des documents "papier" des entreprises sous forme numérique, et leur mise à disposition sur l'intranet ou dans des bases de données, a étendu de manière similaire le volume de données présent sur les réseaux d'entreprise. Si toute évaluation, s'agissant de données privées, est très difficile, leur taille globale est probablement très supérieure à l'internet tout entier. Les réseaux privés de grandes entreprises multinationales peuvent même rivaliser à eux seuls avec le web.

Cette inflation a généré très tôt (dès 1994 sur internet, avec la création du service de recherche de **Yahoo** [@YAHOO]) le besoin d'instruments adaptés de recherche et d'accès aux documents, à l'origine un simple classement des documents dans une hiérarchie de catégories. Un mode de classement similaire s'est fait jour rapidement au sein des intranets, permettant dans les deux cas la création de répertoires spécialisés.

Cependant, de telles classifications ne peuvent prendre en compte qu'un nombre limité de documents. Des hiérarchies trop complexes deviennent inutilisables, et des catégories contenant trop de documents ne sont plus explorables facilement. Ce phénomène a amené le développement d'outils de recherche textuels, non plus axés sur une classification préalable des documents, mais sur une recherche selon des critères de contenu des documents. Ces critères sont spécifiés par l'utilisateur au moment même de la recherche, sous la forme d'une requête précise. Le traitement de cette requête permet de construire une liste de documents en réponse.

Beaucoup plus flexible, et sans limitation de taille de l'ensemble de documents à explorer, ce mécanisme de *text-retrieval* (TR) a été développé pour des bases documentaires privées en 1990, et est apparu dès 1995 sur le web, avec **Altavista** [@AV] et les solutions **Inktomi** [@INK].

Si le système de recherche textuel lui-même n'est pas limité en taille, l'exploitabilité des résultats devient vite aléatoire lorsque des critères trop larges amènent des listes de résultats dépassant la centaine de documents. La plupart des outils de text-retrieval ont donc inclus des méthodes de tri des résultats, donnant un indice de pertinence, explicite mais censé approcher l'estimation qu'en pourrait faire l'utilisateur.

... ou un écheveau ?

Mais la notion même de pertinence d'une information a été largement influencée par l'intégration d'internet comme outil de travail quotidien. Si la réputation de la source d'où provient un document et la fréquence d'apparition des termes de recherche restent de très importantes indications de pertinence, la qualité du réseau de documents et sources auxquels il est relié devient primordiale. Suivant le pionnier en la matière (**Google** [@GOOGLE]), de nombreux outils de recherche intègrent maintenant cette information dans leurs évaluations de pertinence.

Cette assimilation a aussi des répercussions dans la mise en forme de l'information. Les documents de synthèse, technologiques, commerciaux ou financiers délaissent une analyse linéaire pour une interprétation selon des angles multiples, mettant en valeurs les liens entre éléments d'information. Les médias d'information, en ligne ou papier, adoptent de plus en plus une présentation en *grappe*, un groupe de commentaires ou d'analyses autour d'un fait central. Ici, les liens entre documents ne sont plus un indice de pertinence, mais bien l'information elle-même.

A la difficulté de trouver l'information pertinente, s'ajoute donc la nécessité de permettre une conceptualisation complète et une navigation aisée dans de telles grappes. Les outils internet actuels (en particulier le format HTML) trouvent ici leurs limitations. C'est en particulier pour résoudre ce type de problèmes que de nouveaux formats sont en cours d'élaboration (voir par exemple les travaux du W3C pour le XHTML [@XHTML]).

1.2 Extraire beaucoup / bien, problématique dépassée

Les outils classiques d'extraction de connaissances, et en particulier de text-retrieval, doivent trouver un délicat équilibre entre quantité et qualité des informations extraites. Si l'on privilégie la qualité, on peut choisir de n'extraire que des documents correspondant parfaitement au type recherché, au risque de ne pas fournir des documents qui auraient satisfait l'utilisateur. Au contraire, il est possible de retourner toutes les informations pouvant être en rapport avec le but recherché, mais on peut dans ce cas perdre l'information pertinente dans une masse d'éléments sans intérêt.

Eviter ce dilemme ne peut se faire qu'en définissant un objectif de recherche plus précis. Diverses approches ont été tentées (raffinage, sémantique, ...), sans pour autant résoudre complètement ce problème[4]. En réalité, la méthode de définition d'objectifs de recherche textuelle repose sur l'utilisation de termes (mots, symboles, identifiants) dont la signification n'est pas immuable, tant dans leur usage dans les documents, que dans leur emploi par un utilisateur lors de la formulation d'un objectif de recherche. Un outil de recherche générique ne peut dès lors que fournir des résultats dont la précision est limitée par la variabilité des interprétations de ses utilisateurs.

Pourtant, dans un réseau d'informations de plus en plus profondément reliées entre elles, la difficulté n'est plus tant de trouver l'information recherchée, que de relier les différents éléments présentés de manière à permettre à l'utilisateur de les utiliser efficacement. On se heurte ici à un problème de précision et de dépendance à l'utilisateur encore plus grand que pour la recherche elle-même. Car si un certain consensus existe quant à l'usage de la langue, les méthodes de raisonnement et d'analyse de deux utilisateurs peuvent être orthogonales, non seulement à cause de modes de pensée différents, mais aussi par nécessité : dans le domaine médical, un biochimiste ne choisira pas le même type d'analyse de l'information qu'un clinicien, par exemple.

1.3 Les exigences d'un grand web, au sens de réseau de liens

La banalisation d'internet a amené un bouleversement des modes d'usage et d'analyse de l'information. Malgré des techniques très élaborées, l'utilisation de matériel puissant (des index de plusieurs centaines de térabits et des bandes passantes supérieures à 10 Gb/s pour les grands moteurs de recherche internet),

[4] Il ressort des "Search Engine Meeting" de 2000 et 2003 que même les solutions les plus récentes ont des capacités de discrimination insuffisantes pour satisfaire les utilisateurs [SEM5, SEM8].

la spécialisation dans des domaines particuliers, les outils de recherche d'information, qu'ils soient grand public ou professionnels, ne parviennent plus à gérer la taille de ce domaine, et la complexité des attentes des utilisateurs.

Ceux-ci n'attendent plus seulement d'un outil de recherche une simple "extraction" de documents pertinents, mais une véritable aide à la navigation, amenant à la fois un cadrage fin d'une portion de la base documentaire explorée, fournissant l'information qu'ils recherchent, et une présentation adaptée à l'utilisation qu'ils vont en faire.

De nombreuses équipes travaillent sur des modes de visualisation de données organisées en grappe adaptés à l'exploration par des utilisateurs [HagSte99, HagSte00, Qui00]. Encore limités par les moyens matériels couramment disponibles, ils offrent cependant des possibilités très prometteuses. En revanche, les études sur des moyens de text-retrieval personnalisés et adaptatifs sont encore peu nombreuses, et c'est sur cet aspect que nous avons axé notre recherche.

2 Objectif de la recherche

2.1 Profiling, text-retrieval et apprentissage

Déterminer explicitement l'ensemble des informations pertinentes permettant de décrire le comportement d'un utilisateur d'outils de text-retrieval est très difficile. Il s'agit essentiellement de données floues, d'acquisition délicate. L'ensemble des dérives de sens personnelles associées à certains termes, par exemple, est une donnée importante mais qui nécessiterait d'interroger longuement l'utilisateur, pour obtenir des informations ayant une valeur individuelle réduite. On devrait aussi prendre en compte des notions dont la formulation même est malaisée, comme la présentation la plus adaptée au mode de travail de l'utilisateur. Formaliser et acquérir ces informations a retenu l'attention de nombreux chercheurs, et on pourra par exemple consulter [SolCra98, BucBau+99, FieRog00].

Dans tous les cas, il est nécessaire de se doter au préalable d'un *modèle* de l'utilisateur, sous forme d'un nombre fini mais probablement grand de variables censées décrire complètement (au moins pour les besoins du système) tout utilisateur. Il faut aussi savoir traduire ce modèle en procédures de recherche. Lorsqu'ils reposent sur des éléments simples, relatifs aux besoins fonctionnels d'un utilisateur (type de formation, domaine de compétences, rôle...), ils peuvent être aisément traduits mais ne permettent pas d'accéder à une réalité nuancée. Lorsqu'ils sous-tendent des hypothèses psychologiques élaborées, ils sont en mesure

de fournir une analyse fine, mais l'implémentation des outils logiciels correspondant est très difficile. De plus, toute faiblesse du modèle rend l'outil inexploitable.

D'autres approches permettent de se passer de "modèles" aussi encombrants. Dans des contextes où les documents sont par ailleurs organisés en catégories, l'analyse statistique des précédentes recherches permet d'augmenter considérablement la sélectivité. Si cette technique ne s'applique qu'à un contexte documentaire simple, elle est bien éprouvée et déjà utilisée à l'échelle industrielle[5].

Il reste que ce système est encore insuffisant pour prendre en compte des recherches très pointues, sur un ensemble de documents ne possédant pas une catégorisation simple. C'est par exemple le cas de bases de publications scientifiques et de brevets[6], ou les bases privées de grandes entreprises.

Nous tentons de répondre à cette attente en personnalisant la réponse des moteurs de recherche textuels dans de telles bases, par l'élaboration de profils utilisateurs déterminant la manière dont est construite cette réponse. Grâce à des techniques incrémentales de construction des profils, que l'on peut voir comme un apprentissage, nous cherchons à maximiser la satisfaction de l'utilisateur dans les résultats renvoyés. L'observation et l'analyse des comportements de l'utilisateur en réaction aux résultats fournis nous permet d'évaluer cette satisfaction, et de diriger l'optimisation des profils.

2.2 Interaction et Algorithmes évolutionnaires.

Fournir une mesure explicite de la performance des outils de recherche est difficile, et fait l'objet de nombreux travaux (présentés notamment aux conférences TREC [@TREC]). Cependant, du point de vue des utilisateurs, la notion d'objectif de recherche est souvent floue. La notion même de résultat satisfaisant dépend de leur niveau préalable d'information (un document introductif pourra être un bon résultat pour un novice et un mauvais pour un spécialiste).

La seule mesure réellement adéquate provient en fait des utilisateurs eux-mêmes, seuls capables de déterminer si un document particulier répond ou non à leur attente. Cependant, demander trop d'informations à l'utilisateur rend pénible la manipulation du système. Et une telle évaluation humaine est par essence

[5] On pourra citer l'exemple d'**Amazon** [@AMAZON] : si pour des raisons de secret commercial peu d'informations ont été publiées concernant le fonctionnement de leur système de recherche, l'analyse de l'évolution des réponses apportées en fonction des actes de recherche permet d'en déduire que ce type de méthodes a été employé dans la conception de ce système

[6] C'est un des principaux problèmes rencontrés par Novartis lors de l'exloitation de bases référençant la littérature scientifique, comme **EmBase** [@EMBASE] (base médicale et pharmacologique de domaine public) et auxquels tente de répondre le logiciel Ulix [VacPar+01]

sujette à imprécision et variabilité. L'apprentissage des profils devra donc se faire dans un contexte d'informations incertaines et parcimonieuses.

Un autre obstacle est le peu de connaissances disponibles sur les objets à apprendre. Ne pas tenter de modéliser l'utilisateur apporte une plus grande souplesse. Mais l'absence de modèle conduit à des espaces d'apprentissage sur lesquels on dispose de peu d'informations. Ce qui signifie, si l'on utilise le vocabulaire de l'optimisation, que la topologie de l'espace de recherche est mal connue.

Nous avons étudié la mise en œvre d'algorithmes évolutionnaires pour réaliser cette optimisation difficile. Il présentent en effet plusieurs caractéristiques permettant de répondre à ce type de problèmes.

Les algorithmes évolutionnaires, inspirés de la théorie darwinienne, ont montré dans des contextes variés leur capacités d'optimisation dans des contextes extrêmes. Les outils dont ils se servent pour parcourir l'espace de recherche sont simples, et peuvent être construits empiriquement en conservant de bonnes performances. Ils sont peu sensibles à un bruit appliqué à l'évaluation utilisée pour conduire l'optimisation, et l'essentiel de leur coût algorithmique réside le plus souvent dans cette évaluation. Enfin, ils offrent un nombre important de points de contrôle, permettant d'adapter le système aux spécificités des bases et environnement linguistiques.

2.3 Structure de ce document

Après cette introduction, nous commencerons par rappeler les différentes techniques aujourd'hui utilisées dans les quatre domaines principalement concernés par cette recherche : analyse sémantique, text-retrieval, outils intra/internet, et algorithmes évolutionnaires. Nous abordons essentiellement les outils – méthodologies, structures de données ou algorithmes – auxquels nous avons fait appel, et leurs alternatives éventuelles, ainsi que les éléments théoriques fondant leur utilisation.

Nous examinons ensuite les contraintes pesant sur l'association et la structuration de ces outils pour permettre l'optimisation de profils de recherche, et les solutions apportées, ayant abouti à la création d'un prototype.

Nous consacrons une troisième partie au paramétrage de ce système, tant du point de vue du fonctionnement interne que de la qualité des ressources qui doivent lui être fournies. Nous évoquons aussi les méthodes d'évaluation de sa performance, liée à la volatile notion de "satisfaction utilisateur".

Enfin, nous envisageons les –très nombreuses– perspectives de recherche ou-

vertes par les résultats que nous avons obtenus, notamment dans le domaine des algorithmes génétiques, et des contextes d'utilisation des outils de recherches.

Chapitre 2

ETAT DE L'ART

Résumé du Chapitre.

Les solutions de TR sont des outils souvent complexes et volumineux. C'est le cas, de manière générale, de tous les outils de manipulation du langage. Ceux-ci doivent faire appel à des outils et ressources sémantiques permettant de traiter les structures du langage naturel, beaucoup plus lâches que les structures informatiques classiques. S'ajoute à cela, dans le cas du TR, la nécessité de tenir compte, à une extrémité de la chaîne de traitement, de la taille et de la richesse des bases documentaires, et à l'autre extrémité de l'expertise des utilisateurs.

L'interaction avec ceux-ci se fait le plus souvent au sein d'un modèle client-serveur (sur le modèle d'internet). Si ce mode d'interaction répond aux impératifs de performance, il limite les informations échangées et leur persistance. Il est alors nécessaire d'apporter des solutions spécifiques aux questions d'authentification, de stockage des données, et de chronométrage des interactions, indispensables pour envisager la personnalisation de l'outil de TR au moyen de profils utilisateurs.

Les Algorithmes Evolutionnaires, avec lesquels nous avons choisi de réaliser l'optimisation des profils, sont une méthode d'optimisation fondée sur le modèle darwinien de l'évolution naturelle. Ils ne nécessitent qu'une connaissance très limitée de l'espace de recherche, et permettent de traiter l'optimisation de fonctions bruitées ou mesurées de façon approximative.

L'approche parisienne des AE consiste à représenter une solution au problème par l'ensemble de la population manipulée. Celle-ci s'est révélée plus économique que l'approche classique lorsque obtenir une évaluation de la fonction à optimiser est coûteux, comme dans le cadre d'AE interactifs.

1 Outils textuels et text-retrieval

Cette recherche se situe à la croisée de quatre domaines : text-retrieval, sémantique, protocoles internet et enfin algorithmes évolutionnaires. Les technologies de ces différents domaines évoluent très rapidement, et les systèmes les mettant en œvre sont souvent incompatibles. Qui plus est, leur coût en temps de calculs en en espace de stockages est généralement important.

Pour permettre la réalisation efficace de tests, et l'implémentation d'un prototype ayant des performances acceptables, nous avons employé des technologies parfois relativement anciennes. De mise en œvre simple, de coût algorithmique modéré, elles nous permettent en plus de disposer d'éléments de comparaison éprouvés.

1.1 Bases documentaires

La forme la plus simple d'une base documentaire est une simple collection de fichiers texte. C'est d'ailleurs la forme que prennent la plupart des bases de test, voir paragraphe 1.3. Mais la gestion de bases contenant plusieurs millions de documents, comprenant texte, mots clés, schémas et graphiques, voire des données multimédia, appelle l'utilisation d'outils spécifiques pour accéder aux documents, pour des raisons de performance autant que pour permettre une consultation appropriée de documents complexes.

Ces outils sont soit dédiés au stockage matériel des données, le but recherché étant alors la performance et la fiabilité, soit destinées à permettre un accès simple à ces base étendues, visant alors l'ergonomie de la consultation. Souvent intégrés, ces deux aspects font cependant appel à des technologies largement indépendantes : systèmes de stockage et d'archivage d'une part, systèmes de gestion de contenus documentaires d'autre part.

Stockage, formats et structure

Les bases documentaires accessibles par internet sont le plus souvent dans le format le plus populaire sur le web : le HTML. Les images, éventuellement adjointes à ces documents, sont souvent elles aussi stockées dans ces bases pour assurer un rendu graphique optimal des documents.

En ce qui concerne les bases privées des entreprises, le paysage de formats est beaucoup plus diversifié. Des documents en texte simple sont souvent extraits de documents papier par des logiciels d'OCR (Optical Character Recognition, ou

reconnaissance de caractères). Une partie des documents est ici aussi en HTML, provenant de l'intranet d'entreprise, ou de la conversion de documents simples dans d'autre formats.

Si ce format fournit un rendu graphique adéquat, les "balises" HTML, (permettant d'indiquer le type de rendu souhaité pour une portion de texte) apportent peu d'informations en termes de sens et de structuration des documents.

Cependant, la plus grande partie des documents est généralement dans des formats plus complexes, et difficilement traduisible de l'un à l'autre : LaTeX (dans le cas de bases scientifiques), ou plus souvent encore formats propriétaires (**Word**, **Excel**, **Powerpoint** parmi les plus courants) nécessitant des logiciels particuliers pour leur exploitation. Ils comportent parfois des références particulières à un ensemble de mots clés, un résumé, des références bibliographiques ou lexicales en relation avec certains termes, et bien d'autres informations spécifiques, localisées dans le document ou non.

Une uniformisation de ces formats se fait jour très progressivement grâce à XML. Les indications présentes dans la plupart des formats propriétaires peuvent le plus souvent être incluses dans une traduction XML au moyen de tables de description *ad-hoc* pour chaque format (DTDs, établis suivant les spécifications de la norme XML, voir la recommandation W3C [XML-1.0], et plus généralement [@XML]).

Indépendamment du format, ces documents sont généralement rendus accessibles via des serveurs web dédiés, sauf dans le cas de documents sonores ou audiovisuels, nécessitant des outils spécifiques (mais ne rentrant pas non plus dans le champ d'application du text-retrieval).

Un cas très particulier, mais courant, est celui des *bases de données relationnelles* (plus connues sous leur acronyme anglais RDB), pour lesquelles le mode de stockage ne peut être dissocié du format. Permettant d'accéder à des données très catégorisées, par domaine, table, champ, elles permettent de constituer des listes d'objets possédant un certain nombre d'attributs. De manière simplifiée, les champs de chaque table sont les valeurs de ces attributs.

Les systèmes de RDB et les sources XML ont actuellement tendance à converger rapidement (voir [@RDB]), qu'il s'agisse d'outils logiciels ou de formats de bases, car le pouvoir de description entre une table de RDB et un DTD pour XML sont très similaires. En revanche, les performances des RDB restent nettement supérieures, particulièrement lorsqu'il s'agit de répéter un grand nombre de fois la même opération sur un grand nombre d'objets (ou de documents).

Les entreprises tentent aujourd'hui de standardiser autant que possible leurs

formats de documents destinés à la publication interne, à l'origine pour en faciliter le traitement automatique. Lorsque des standards sont établis pour une proportion suffisante de documents, il devient possible de définir des DTD permettant de les convertir en XML, ou, approche préférée lorsque ces documents comportent des données techniques ou financières, de les intégrer dans une RDB.

Dans les deux cas, cette transformation fournit aux outils de TR une collection de champs annotés pour chaque document, permettant de réaliser des recherches selon des critères fins.

Indexation, accès et DB

Qu'il s'agisse de texte simple ou possédant une mise en forme graphique, organisé ou non en catégories, l'accès à chaque élément d'une collection de documents "plats" (i.e. sans base de donnée associée) nécessite une méthode d'identification unique de cet élément. Sur internet, il s'agit le plus souvent d'un URI (Unified Ressource Identifier), très similaire au chemin d'accès dans un système de fichier, utilisés dans les bases stockées localement (par exemple sur un disque dur). Il s'agit de toute manière d'une chaîne de caractères, parfois assez longue, référençant de manière biunivoque un document pour le système d'accès.

Cependant, du point de vue de l'utilisateur, ces identifieurs sont peu compréhensibles, et ne permettent pas de retrouver aisément un document particulier. Assurer un accès aisé pour les utilisateurs passe de manière inévitable par une *indexation*. Il s'agit de constituer une table croisée d'identifieurs et d'indications plus "parlantes". Le contenu précis de ces indications dépend bien sûr du moteur de recherche utilisé pour parcourir ces index (voir section 1.2), mais il s'agit généralement de mots extraits des documents.

L'organisation pratique de la table est conditionnée par les performances algorithmiques recherchées, et la quantité de données à classer. Les techniques de tri rapide ($\mathcal{O}(n\log(n))$) permettent de construire des index pour lesquels les temps de recherche sont logarithmiques. Les méthodes à base de tables de hachage, d'accès plus rapide encore, sont en revanche à réserver aux index qui ne doivent pas être reconstruits souvent : la fabrication de ces index, ou leur mise à jour dans le cas le pire est longue.

Lorsque la base documentaire est une RDB, la création d'un tel index n'est pas nécessaire. En effet, s'il est possible d'accéder à une propriété particulière d'un objet au moyen d'un identifieur unique, l'accès au contenu des RDBs se fait au moyen d'un langage spécifique (le SQL, Simple Query Language [SQL]), ou

de dérivés similaires, particuliers à un format de RDB. Il fournit à la fois les outils permettant des accès extrêmement efficaces à des séries de champs, mais aussi un moyen de parcours performant des tables de la base, faisant double emploi avec un index externe. En fait, les performances obtenues sont suffisantes pour justifier le stockages des index de bases ne comportant que des documents plats sous forme de RDB.

Cependant, qu'il s'agisse d'index ou de RDB, ces outils ne sont pas adaptés à une utilisation directe, et ne forment qu'une base pour le développement d'outils d'exploration et de recherche.

1.2 Moteurs de recherche et text-retrieval

Du point de vue de l'utilisateur, les moteurs de recherche se distinguent par la manière de formuler un objectif de recherche, et la manière de présenter les résultats.

La présentation des résultats est encore, pour la plupart des moteurs disponibles sur internet, une liste de liens pointant vers les documents répondant à l'objectif, chacun accompagné par quelques informations sur le document et, éventuellement, un court extrait ou résumé. Pour des utilisations spécialisées, d'autres modes de visualisation des résultats ont été développés : navigation sémantique, recherche hiérarchique, clustering (voir par exemple : [VacPar+01]).

En revanche, l'évolution de la richesse du langage de requête et de la complexité de son interprétation ont été à l'origine de la plus grande part de l'amélioration des performances du TR au cours des dix dernières années.

Recherche booléenne

L'outil le plus simple de TR est aussi vieux qu'UNIX : il s'agit de *grep*. Il suffit parfois d'en connaître un mot pour retrouver un texte sur un sujet donné dans une collection de documents. Par exemple, rechercher "prévisions météo" dans le contenu des nouvelles du jour nous fournira très probablement la bonne page. Si les résultats sportifs sont un peu plus difficiles, car il y a bien des façons de dire que l'on parle des résultats du match de la veille, demander les documents contenant les mots "match", "résultats" et les noms des équipes fonctionnera généralement bien.

C'est une formalisation de ce procédé empirique qui est à l'origine des outils de recherche booléens. Le langage de requêtes est une formule logique, construite

à partir d'opérateurs booléens (d'où le nom de ce type de formulations), unaires ou binaires **#NOT**, **#AND**, **#OR**. Les terminaux sont des termes (chaînes de caractères). Cette formule logique est évaluée sur chaque document de la base, la valeur d'un terminal étant **vrai** si et seulement si le terme qu'il représente est présent dans le document. Une requête correctement formulée permet donc d'identifier – et de retourner à l'utilisateur – exactement les documents souhaités.

Ces formules se traduisent de manière ensembliste. En remplaçant chaque terminal par l'ensemble des documents qui le contiennent, et les opérateurs **#AND** par une intersection d'ensembles, **#OR** par une union, **#NOT** par un complément dans l'ensemble des documents de la base, on obtient l'ensemble des documents à retourner.

Si du point de vue de l'utilisateur cette représentation est anecdotique, elle est probablement la principale raison du succès de cette méthode de formulation du point de vue de l'implémentation. Evaluer une telle formule logique sur tous les documents de la base est extrêmement coûteux, d'autant plus que le nombre de requêtes est important. Cependant, si l'on limite les terminaux acceptables à un sous-ensemble fini de chaînes, et que l'on en crée un index (voir 1.1), construire l'ensemble des documents contenant un terme donné est algorithmiquement économique. Et à partir de ce point, déterminer l'ensemble des documents validant la requête est une simple suite d'intersections, d'unions et de compléments, opérations elles aussi peu coûteuses. Le coût est par ce biais concentré dans la construction des index, ce qui ne doit être fait qu'une fois (éventuellement, lors de chaque modification de la base), et non à chaque requête.

Des règles pratiques dirigent la détermination des termes acceptés. Ce sont des mots, définis dans toutes les langues comme une suite de caractères alphanumériques, et délimités par des espaces ou caractères de ponctuation. La seule ambiguité, mineure, réside dans le caractère "-", qui dans certaines langues peut entrer dans la composition d'un mot. Sont de plus éliminés les mots trop courts, qui n'ont souvent pas de signification propre (par exemple, en français, déterminants et mots de liaison), permettant d'alléger l'index et donc de réduire le coût de sa production.

Recherche booléenne floue à base de vocabulaires

Ce système d'indexation des documents par les mots qu'ils contiennent, pris sans modification, est mis en défaut lorsque la graphie des mots varie en fonction de leur environnement grammatical : accords, conjugaisons ou déclinaisons. De

ce fait, la recherche d'un mot donné peut échouer, alors que ce mot se trouve effectivement dans un document, dans une forme dérivée. Ainsi, pour reprendre l'exemple du paragraphe précédent, rechercher "*résultats*" peut échouer parce que ce terme n'est présent que sous sa forme singulière "*résultat*". En effet, les opérations ensemblistes sur les index décrites plus haut se basent sur une identification *exacte* des entrées de l'index, pour laquelle "*résultats*" est une entrée distincte de "*résultat*" (voir un développement de ce problème dans [Par98], et 6.2.3).

Une manière de contourner ce problème, efficace dans la très grande majorité des cas, est de transformer ces entrées multiples en une seule. Celle-ci est appelée par les linguistes la forme lématique, ou lemme, du terme, et correspond à diverses conventions en fonction des langues. Pour les noms et adjectifs, il s'agit de la forme singulière masculine. Pour les verbes, la forme infinitive, ou bien encore la troisième personne du singulier du présent. Les adverbes se passent généralement de convention.

Cette définition n'est cependant qu'une convention linguistique, et ne fournit pas d'algorithme permettant de déterminer réellement cette forme (voir par exemple [EynJak+00], [Kin99]). Deux techniques sont employées. La première, et la plus sûre, repose sur l'emploi d'un dictionnaire, donnant chaque forme possible de chaque lemme. Ceci est évidemment très coûteux en espace de stockage et en temps de recherche, mais encore plus en investissement humain lors de la constitution du dictionnaire, et n'est de toute manière praticable que pour un vocabulaire limité. Mais cette méthode assure des résultats exacts.

On peut s'abstraire de la nécessité de tels dictionnaires en utilisant des heuristiques de lemmatisation. C'est la méthode de loin la plus employée, avec des formules de qualité et précision diverse. Une heuristique simple, applicable aux noms et adjectifs, est par exemple de supprimer tout suffixe "*s*" ou "*es*". Une courte liste d'exceptions permet d'éviter les faux-pas. Un ensemble d'heuristiques complet et raisonnablement fiable, pour un dictionnaire d'environ 60000 mots, demande un investissement humain important, mais est réalisable, tandis qu'un lexique complet des formes accordées pour un tel ensemble de termes ne le serait pas.

Ces deux méthodes partagent un inconvénient important et insoluble : elles ne sont valables que pour une seule langue. Tant les heuristiques que le dictionnaire de base sont à reconstruire lorsqu'on change de langue, ce qui rend impossible leur utilisation dans un contexte multilingue, par exemple pour le web entier. Dans ce cas, il est nécessaire de trier au préalable les documents selon leur langue. Par exemple, à l'aide d'une identification statistique, en comparant les

mots du document (les chaînes sans espaces) aux formes tirées des dictionnaires ou heuristiques disponibles pour chaque langue. Mais cette méthode échouera de toute manière dans le cas d'un document multilingue, par exemple une publication dans laquelle a été inclus un résumé dans une autre langue.

Recherche sémantique : expansion, thesauri, et coût algorithmique

Si le problème des différentes graphies d'un terme peut être résolu par des moyens purement algorithmiques, celui des différents moyens d'expression d'une notion, autrement dit le problème de la synonymie, est largement plus complexe. Un exemple élémentaire : le mot "oignon". L'horticulteur recherchant un document soutenant ce thème souhaitera probablement que soit listés aussi ceux contenant le terme "bulbe", tandis que le gastronome pourra vouloir voir apparaître aussi les documents contenant le terme "échalote". Au-delà de l'ambiguité entre ces deux aspects (bulbe de fleur / légume), il n'est évidemment pas possible de réaliser une telle correspondance à l'aide d'un algorithme. Il est nécessaire de constituer une table de correspondances, ou encore un dictionnaire des synonymes.

Ces dictionnaires sont en fait des *thesauri* (voir [SalMcG83, Fos97]). La relation de synonymie n'étant pas transitive, il s'agit de structures plus complexes qu'une simple table de correspondance, introduisant une notion complémentaire de celle de terme : celle de sens (voir section 2.1). Dans l'exemple ci-dessus, on effectue de fait un choix entre deux sens. Le nombre de sens et le nombre de mots d'un thesaurus sont généralement du même ordre de grandeur. Ce concept impose de voir sous un nouveau jour tant la phase d'indexation de la base documentaire que l'interprétation des requêtes.

Deux choix s'ouvrent alors à nous, selon la chronologie de l'utilisation des thesauri. Si l'on décide de les utiliser essentiellement lors de l'indexation, l'index par termes se transforme en index par sens. Chaque sens est identifié par un simple numéro, ou par un "représentant principal", constitué d'un terme prédominant dans ce sens accompagné d'un numéro si le terme choisi a lui-même plusieurs sens. La recherche est alors une recherche par sens, chaque terme entré étant remplacé lors du traitement de la requête par le sens associé. A l'opposé, il est possible de conserver un index par termes, mais le traitement des requêtes s'accompagne d'une phase d'*expansion sémantique*, visant à transformer chaque terme en une liste de synonymes (voir en particulier [Voo94]).

Un des éléments de décision entre les deux méthodes d'indexation est le coût algorithmique. Réaliser une expansion pour chaque terme d'une requête est

assez peu coûteux, les requêtes contenant généralement un nombre réduit de termes (inférieur à la dizaine). Au contraire une indexation par sens nécessite de faire une recherche du sens associé à chaque mot de chaque document, ce qui pour des bases typiques (1M de documents, 50 pages en moyenne) demande de l'ordre de 10^{10} appels. Cependant, pour des "gros" serveurs de recherche, dont la fréquentation quotidienne est importante, ce coût très important peut s'avérer inférieur au coût cumulé des expansions de requêtes, et surtout peut être contrôlé temporellement et réparti entre plusieurs machines.

Cette description simplifiée fait abstraction de la principale difficulté pratique de mise en oeuvre des thesauri. Si le passage des sens aux termes est aisé, puisque l'on peut généralement se contenter de la liste de *tous* les termes associés, l'inverse – associer un sens à un terme – fait intervenir une étape de désambiguation : déterminer quel sens, exactement, doit être utilisé pour ce terme. On l'a évoqué plus haut, à propos du mot "oignon", un terme pouvait évoquer plusieurs sens, il est nécessaire de se servir du contexte pour informer le choix de sens. Or il n'existe pas de méthode générale, à l'efficacité éprouvée pour toutes les langues. Un des outils les plus efficaces dans le cas général repose sur une analyse statistique des triplets de termes. Ses performances ont été montrées dans le cas de la correction d'orthographe (avec une étude des triplets de lettres), et a été testée avec succès dans des environnements multi-linguistiques (voir [EveRot+99, MasTuf97]), mais est ici mise en échec par plusieurs langues dont les règles grammaticales éloignent les éléments signifiants (par exemple l'allemand, avec le rejet du verbe en fin de phrase). De plus, les informations contextuelles sont parfois absentes dans les documents à indexer, et pratiquement toujours dans les requêtes.

Dans la plupart des implémentations actuelles, une solution remédiant partiellement à ce problème détermine le choix d'une indexation par mots, au-delà des considérations de coût, puisqu'elle permet de réaliser une expansion affaiblie, évitant les choix de sens et utilisant pour chaque terme tous les sens disponibles.

1.3 Outils d'évaluation

Indices de performances du text-retrieval

Considérons une base documentaire, B, dans laquelle nous allons rechercher les documents correspondant à un certain concept C. Un moteur de recherche idéal μ (ou un expert, ayant le temps d'explorer tous les documents de la base) obtiendrait un ensemble de documents $S = \mu(C)$, nommé *ensemble objectif*. Le moteur de recherche dont nous disposons, $\tilde{\mu}$, imparfait, fournit un ensemble de résultats $R = \tilde{\mu}(C)$. L'ensemble des "bons" résultats dans cette liste est

$D = R \cap S$.

On définit à partir de ces ensembles (fig. 2.1) deux quantités servant à mesurer la performance d'un moteur de recherche, à partir d'une base de tests contenant des documents et les ensembles objectifs correspondant à une liste de requêtes types (suivant en cela [CleKee66]). La première est la *précision*, $p = \frac{|D|}{|R|}$. Elle marque la proportion de documents listés à juste titre comme répondant à cette requête. La seconde est le *taux de rappel*, $r = \frac{|D|}{|S|}$. Elle marque la proportion de documents listés parmi la totalité des documents qui auraient dus être retournés.

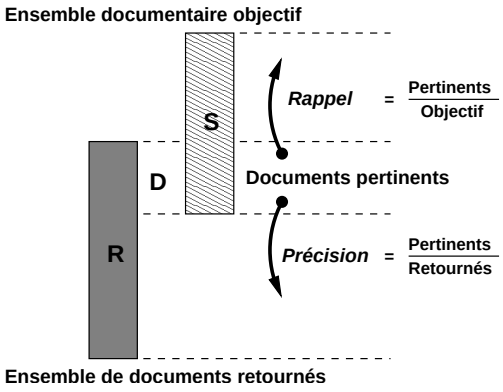


FIG. 2.1 – Taux de rappel et précision.

Ces deux quantités peuvent être définies soit pour une requête précise, soit pour un ensemble de requêtes de test. Dans ce dernier cas, les ensembles cités plus haut sont la réunion des ensembles correspondants pour chaque requête.

Notons que ces notions ont leur équivalent en théorie de l'apprentissage : $R \backslash D$ sont les faux-positifs, $S \backslash D$ sont les faux-négatifs. Ce qui fait de ces quantités –outils de mesure de qualité standards pour la communauté du text-retrieval– des éléments d'évaluation essentiels dans l'apprentissage de profils, comme nous le verrons au chapitre 3.

Contenus des bases de test en TR

Les bases de tests ("benchmarks") utilisées en TR ont généralement pour objectif de fournir des évaluations numériques, reproductibles, des outil de TR, permettant ainsi de les comparer de manière objective. Leur contenu est déterminé en fonctions de ces objectifs. Tout d'abord, il faut disposer d'un ensemble de documents homogène et de qualité suffisante (ni trop courts, ni trop vo-

lumineux, et possédant une cohérence linguistique et sémantique). Il est aussi nécessaire de pouvoir déterminer la validité d'un résultat de requête particulier. Ceci est réalisé en associant à chaque document des informations qui ne sont pas utilisées pour le TR, par exemple une liste de mots-clés, l'appartenance à une série de sujets prédéterminés, ou des informations sémantiques construites par des experts humains, par exemple une liste des n plus importants sujets abordés dans chaque document.

Le modèle des **TREC**, qui est un des standards du domaine (voir [@TREC], et [KimZha+00, ShiKim+99]), montre que cet objectif peut être atteint avec robustesse et simplicité. Durant plusieurs années, les bases de tests ont comporté un ensemble de documents relativement réduit (de l'ordre du millier de documents), une liste de requêtes types exprimées en langage naturel, et pour chaque requête une liste de documents indiqués comme répondant à cette requête par des experts du domaine concerné par l'ensemble documentaire. Eventuellement, ces listes comportent aussi des indices de pertinence pour chaque document. A partir de ces données, il est très aisé d'évaluer taux de rappel et précision pour un moteur de recherche particulier, modulo la traduction de chaque requête dans la forme appropriée pour le moteur (opération introduisant un bruit assez limité si les requêtes sont formulées clairement et simplement).

2 Vocabulaires, sémantique et thesauri

2.1 Réseaux sémantiques

Nous avons mentionné plus haut, sans définitions, plusieurs notions relatives aux outils et bases sémantiques. Nous allons dans cette partie donner une définition formelle de ces concepts.

Notions sémantiques

L'utilisation d'outils sémantiques, par opposition aux outils textuels, repose sur la distinction entre signifiant et signifié, autrement dit entre les mots utilisés et ce qu'ils désignent. L'emploi d'une terminologie particulière permet cette distinction.

Mot : Utilisé en indexation, pour signifier une chaîne sans espaces, cette notion est peu utilisée en sémantique, à cause de son caractère ambigu, en particulier en ce qui concerne les mots composés. Son acception courante entre

en effet en conflit avec l'unité élémentaire utilisée en sémantique, qui est l'entité signifiante, définie indépendamment de la graphie.

Terme : Il s'agit d'un élément signifiant atomique, éventuellement composé de plusieurs mots. Autrement dit, un groupe de mots désignant un objet ou une idée ne pouvant être défini autrement que par une périphrase. Par exemple, "document" et "transcriptase inverse" sont des termes.

Lemme : Forme standardisée unique d'un terme, par exemple la forme singulière masculine, ou l'infinitif des verbes. Comme on l'a vu, elle est utilisée lors de l'indexation, mais sert aussi d'entrée de référence dans toutes les structures relationnelles utilisées en sémantique.

Sens ou **Concept** : L'objet ou l'idée elle-même, par opposition à l'élément lexical utilisé pour le désigner. Si l'on peut voir un terme comme le contenant, ou le véhicule, le sens en est le contenu.

Si l'on accède aisément à des ensembles de termes (phrases, documents...), l'espace de travail le plus pertinent est celui des sens. Mais réaliser la liaison entre sens et termes pose deux difficultés. La synonymie (*plante* et *végétal* par exemple) et la polysémie (*solide* dans le sens de l'état physique, ou de la résistance) ne permettent pas d'établir de relation injective entre terme et sens ou réciproquement.

Vocabulaires, thesauri, réseaux sémantiques

Vocabulaire : Collection de termes, avec l'indication de leur forme lématique, ou simple collection de ces formes (fig. 2.2).

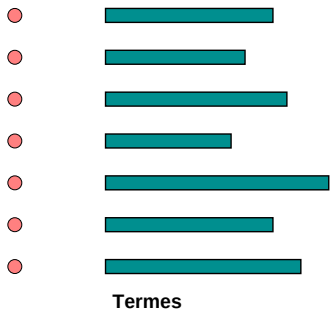


FIG. 2.2 – Vocabulaire : une simple collection de termes.

Thesaurus : Collection de lemmes associés à un graphe de synonymie et parfois à une hiérarchie de concepts (fig. 2.3). Il s'agit en fait d'un "dictionnaire

des synonymes" (voir [Fos97]). Le graphe de synonymie associe deux à deux des termes de sens "proche" (la définition de cette proximité résulte bien sûr d'un *a-priori* linguistique lors de la construction du thesaurus), le plus souvent de manière symétrique. La hiérarchie de concepts formalise des notions telles que "partie de" ou bien "cas particulier de", en fonction du biais choisi pour constituer cette hiérarchie.

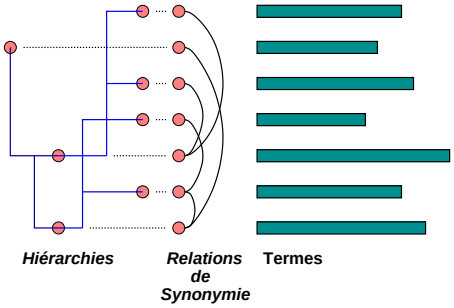


FIG. 2.3 – Thesaurus : graphe de synonymie.

Thesaurus flou : Comme un thesaurus classique, il comporte un graphe de synonymie, mais cette fois les arcs du graphe sont porteurs de poids indiquant le degré d'identité entre l'ensemble des notions véhiculées par chaque terme (voir par exepmle [Bro77]).

Ontologie : Description formelle d'un domaine de connaissance [Gru93]. Le plus souvent, il s'agit d'un ensemble de définitions, liant entre eux les concepts de base du domaine, grâce à une liste de relations fonctionnelles (du type *partie de*, *qualifie*, *est plus grand que*, etc...)[1].

Réseau sémantique : Cette forme très générale de système relationnel basé sur des termes est constitué d'un graphe dirigé annoté dont les noeuds sont des termes (voir les fondements de cette notion dans [ColQui72], et une étude structurelle dans [Bra77]). Une liste de relations est donnée pour chaque type de réseau. Deux termes peuvent être reliés par un ou plusieurs arcs correspondant chacun à une relation dans cette liste (fig. 2.4).

Parmi les types de relation le plus couramment utilisées se trouvent la synonymie, bien sûr, la relation de composition, ou holonymie, "contient" (ou sa duale "est fait de"), la relation de spécialisation, ou hypéronymie, "sorte de", ou encore la relation de similitude, "s'utilise comme". Lorsqu'on traite de réseaux sémantiques liés à un domaine technique, peuvent s'ajouter des relations spécifiques.

[1] Le sens précis à donner à ce terme est contorversé, en particulier en raison d'un usage très différent en philosophie.

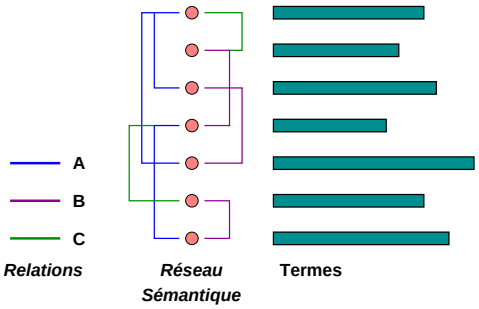


FIG. 2.4 – Thesaurus.

On peut souvent définir dans ce cas, par exemple, la relation de "qualifieur", qui indique qu'un terme peut être adjoint à un autre pour en préciser le sens. Une autre relation utile est celle d'"alternative", qui indique la stricte équivalence de deux termes mais dans des référentiels lexicaux différents, et s'applique à plusieurs formats ou nomenclatures permettant d'exprimer un même objet.

Les thesauri sont une catégorie particulière de réseau sémantique, pour laquelle la liste de relations est très simple : elle comporte la relation de synonymie (dans ce cas, réflexive et symétrique) et parfois, quand existe une hiérarchie de concepts, la relation d'hypéronymie.

Quelques indices simples, issus de la théorie des graphes (voir par exemple [Die00]), permettent de mesurer la qualité de réseaux sémantiques et de la même manière des autres structures dont ils sont une généralisation. Tout d'abord, la première richesse d'un tel réseau est le nombre d'entrées qu'il contient. Plus le nombre de termes qu'il référence est important, moins l'on risque de rencontrer des termes inconnus dans un document ou une requête, ce qui risquerait, entre autres, de fausser une analyse contextuelle. Le deuxième élément important est la connectivité, au sens des graphes, du réseau, définie comme la taille moyenne d'un sous-graphe connexe maximal. Elle représente une mesure de la "granularité" du réseau, permettant lorsqu'elle est élevée de réaliser plusieurs expansions successives pour atteindre le niveau recherché de généralité.

2.2 Identification et extraction

Ces structures (vocabulaires, réseaux sémantiques) permettent une –relative– unification des données d'un outil à l'autre. Leur acquisition reste quant à elle

difficile. Les premiers thesauri, utilisés dès l'époque médiévale[2], reposaient sur le travail de documentalistes, à partir d'ensembles de textes sélectionnés et eventuellement de dictionnaires de langue. Ceci implique une reproductibilité aléatoire (deux termes d'usage similaire pouvaient avoir des listes de synonymes de taille très différente), un faible nombre d'entrées, et une faible connectivité (des listes de synonymes réduites pour chaque terme).

Or la performance des outils de recherche, ainsi que de tous les autres instruments dépendant de ces structures, s'accroît très sensiblement avec la quantité d'information qu'elles renferment. Un grand nombre de termes répertoriés permet l'analyse de documents de forme et de vocabulaire variés. Une grande connectivité (nombre moyen de termes auxquels est relié un terme particulier) fournit un meilleur contrôle sur l'expansion sémantique, et en pratique augmente le taux de rappel. De plus, l'évolution permanente du contenu des bases documentaires, et du vocabulaire lui-même (en particulier dans des contextes scientifiques) impose une mise à jour régulière des bases de vocabulaire.

Aujourd'hui, la plupart des grands thesauri scientifiques sont construits en grande partie automatiquement, et l'intervention humaine se limite à une vérification et une correction des erreurs résiduelles. Les termes, formes lématiques, relations de sens et hiérarchies d'usages sont extraits de la littérature. Toutefois, cette extraction demande des outils informatiques performants, et beaucoup de temps de calcul. Des thesauri comme le MeSH, ou **EmBase** [@MESH, @EMBASE], dans le domaine pharmacie-bio-médical, sont le résultat de plusieurs années de travail, tant pour traiter les masses importantes de publications nécessaires à des thesauri équilibrés que pour vérifier et corriger à la main les résultats obtenus. Les techniques employées doivent être adaptées finement à la qualité des résultats attendus, avec le plus souvent à la clé un équilibre à trouver entre richesse de la base (nombre d'entrées) et finesse des correspondances (hypéronymie et synonymie). Dans tous les cas, quelle que soit l'efficacité des techniques employées, l'adéquation entre thesauri et bases documentaires dépend pour une grande part du bon choix du corpus utilisé pour l'extraction.

Extraction de termes

La construction de vocabulaires (rappelons qu'il s'agit de simples listes de termes, parfois sous leur forme lématique) se base sur des principes similaires aux méthodes d'indexation déjà décrites. Dans la pratique, elle se décompose

[2]Le premier thesaurus de la langue latine construit méthodiquement, le *Glossarium Mediæ Latinitatis* de Charles du Fresne du Cange, date de 1678.

souvent en trois étapes : extraction, sélection, lemmatisation.

La première consiste à répertorier tous les mots présents dans l'ensemble documentaire, et à déterminer les termes qu'ils composent. Cela se fait soit entièrement à la main (sélection qui prend relativement peu de temps, et ne nécessite pas l'emploi de spécialistes), soit par identification des n-uples les plus fréquents.

La sélection consiste ensuite à éliminer les termes non signifiants (éléments grammaticaux, erreurs de frappe, codes d'identification techniques, nombres)[3]. Plusieurs règles président à ce choix. Des éléments objectifs permettent un traitement automatique de la plus grande part des données : mots de moins de deux lettres, comportant une proportion trop importante de consonnes ou des consonnes triplées, trop de chiffres, etc... Des filtres supplémentaires, basés sur la loi de Zipf [Los01], sont souvent utilisés, mais leur efficacité reste très limitée (voir [Li92]). Une vérification manuelle reste dans tous les cas nécessaire, pour s'assurer que des suites de lettres sans signification ne "polluent" pas le vocabulaire.

Extraction lexicale

L'extraction lexicale vise à obtenir dans les textes analysés, non plus de simples mots, mais des termes, clairement identifiés dans leur rôle linguistique. Ce qui implique l'obtention d'informations grammaticales, et la "standardisation" des formes obtenues, en sus de la simple extraction de terme.

Déterminer la classe grammaticale d'un mot nous semble une tâche simple. Mais dans les langues latines, en particulier, où la position des mots est peu contrainte et les variations orthographiques grammaticales (déclinaisons, accords) faibles, automatiser cette analyse repose essentiellement sur des heuristiques. Celles-ci disposent de tables de terminaisons standard permettant d'identifier la catégorie de certains termes, des règles d'exception, etc... Il n'existe pas de procédé systématique valable pour toutes les langues, et la réalisation de ces outils demande donc un investissement important.

Lorsque cette identification est réalisée, les informations obtenues sont utilisées pour faciliter la lemmatisation. Celle-ci résulte souvent du remplacement d'un suffixe (par exemple forme conjuguée d'un verbe) par un autre suffixe standard (forme infinitive, par exemple). Elle repose sur des procédés similaires :

[3]Dans le cas des bases utilisées par Novartis Pharma, on procède en plus à l'élimination des noms de personnes, considérés comme non-signifiants. Les noms commerciaux de produits pharmaceutiques, au contraire, sont conservés et normalisés.

heuristiques, tables d'exceptions et règles de correspondance.

Extraction sémantique

Le but essentiel de l'extraction sémantique est la détermination de correspondances de sens entre termes. Plus que des termes individuels, l'extraction sémantique traite donc des relations apparaissant entre deux termes dans l'ensemble du corpus documentaire.

Si les outils de lématisation utilisés pour l'extraction lexicale sont utiles ici afin d'obtenir un représentant unique de chaque terme, l'extraction proprement dite joue dans ce cadre un rôle moins essentiel. L'analyse des caractéristiques grammaticales peut contribuer à affiner les relations entretenues par deux termes rencontrés dans les mêmes phrases, par exemple par des méthodes statistiques (similaires à celles décrites dans [Tur03]). Mais de telles analyses sont coûteuses sur le plan algorithmique, et leur contribution à la fiabilité des relations sémantiques extraites reste faible. On se contente donc le plus souvent d'utiliser uniquement la proximité de deux termes comme indicateur de leurs relations.

EN TR, cette technique est utilisée principalement de deux manièrs. Pour l'extraction de vocabulaires pertinents, elle vient en complément des techniques décrites précédemment d'élimination des mots parasites. Elle permet alors d'éliminer les termes isolés sur le plan sémantique. Lorsqu'on cherche à établir une véritable carte sémantique de l'ensemble documentaire, il s'agit d'extraire des thesauri. Ce cas, d'importance fondamentale en TR, est détaillé dans la section qui suit.

2.3 Thesauri

Structure des thesauri

Comme on l'a évoqué plus haut, l'élément essentiel d'un thesaurus est le graphe de synonymie. Le graphe d'hypéronymie (il s'agit en fait d'un DAG – graphe acyclique dirigé) est présent dans la majorité des thesauri disponibles commercialement. Cependant son utilisation est moins primordiale, tant dans l'expansion de requêtes que dans l'analyse sémantique de textes.

Il existe de nombreux codages possible d'un tel graphe, dont l'efficacité dépend essentiellement... de l'utilisation que l'on en fait, en d'autres termes du mode de parcours du graphe. Quelques indicateurs numériques sont cependant importants. La "taille" typique d'un thesaurus est de l'ordre de 60000 à 100000

entrées (termes uniques, ou noeuds du graphe). Le nombre de synonymes d'un termes est de l'ordre de la dizaine. L'utilisation la plus fréquente du thesaurus étant la recherche d'une liste de synonymes d'un terme particulier (voir son unique mode d'utilisation, si l'on se limite à de l'expansion de requêtes), de tels chiffres rendent possible l'implémentation de thesauri sous forme de liste de termes portant chacun un vecteur de termes synonymes. Un des premiers thesauri de la langue anglaise réalisés, WordNet ([@WNET]), utilise ce type de stockage (une analyse très complète a été réalisée à ce sujet par les créateurs de cet outil [FelMil98], et constitue encore aujourd'hui une référence du domaine – voir aussi une description très étendue de cet outil dans [Fos97]).

On rejoint dans ce type de formats les index de termes évoqués section 1.1, et les outils employés pour assurer le stockage et l'accès aux thesauri sont souvent identiques (RDB, en particulier, permettant un partage efficace de ces données). On parlera ainsi souvent de "bases de vocabulaire" autant que de "bases documentaires".

Le graphe d'hypéronymie peut lui-aussi être stocké par les mêmes méthodes. C'est aussi le cas des réseaux sémantiques, stockés dans des RDBs (cas rare, puisqu'il en existe peu commercialement disponibles, du fait de la spécificité de cet outil). Ce choix résulte le plus souvent de la familiarité des développeurs et administrateurs avec ces outils, plutôt que d'une réelle nécessité technique.

Modes d'interrogation

Les données contenues dans les thesauri sont essentiellement utilisées selon un mode d'interrogation par terme : il ne s'agit pas de suivre un parcours possédant des propriétés particulières dans le graphe de synonymie, mais au contraire de récolter toutes les informations afférentes à un terme donné. Autrement dit, on cherche à obtenir l'ensemble des synonymes (ou *synset*) d'un terme. Les interrogations sur le graphe d'hypéronymie suivent le même schéma : les informations utiles se limitent à l'ensemble des hypéronymes ou hyponymes d'un terme donné.

Les structures en index et les bases de données relationnelles se prêtent particulièrement bien (tant en terme de coût calculatoire qu'en capacité mémoire) à ce type d'accès. Dans le contexte des RDBs, il s'agit par exemple d'une simple sélection d'une ligne (celle correspondant au terme examiné) suivie de l'accès à une colonne (synonymes, hypéronymes ou hyponymes, selon l'information souhaitée).

Cependant, un tel stockage n'apporte pas de réponse à l'une des raisons

d'être des thesauri : l'identification des formes lématiques. Le très grand nombre de formes que peut prendre un même terme interdit d'adopter une approche exhaustive, revenant à faire une liste de toutes les alternatives. On l'a vu section 2.2, la méthode de lemmatisation la plus utilisée repose sur des heuristiques. Dans le contexte des thesauri, l'interrogation par terme permet d'ignorer totalement d'éventuelles erreurs de ces heuristiques, puisqu'on ne recherche pas une forme exacte (grammaticalement valable, et acceptable pour un humain), mais une entrée dans une liste. Cependant, ces heuristiques dépendent de manière importante de la classe grammaticale des termes concernés. Aussi divise-t-on les listes de termes en catégories grammaticales, permettant une utilisation efficace des heuristiques, et facilitant les tâches d'analyse sémantique basées sur ces thesauri.

Fusion de vocabulaires

Les thesauri utilisés en TR sont le plus souvent construit autour de vocabulaires précis et spécialisés. Les thesauri commerciaux s'adressent à des publics ciblés, scientifiques ou techniques, qui ont besoin d'une topologie très détaillée du vocabulaire de leur discipline, et recherchent donc des thesauri spécialisés de celle-ci. C'est le cas aussi des thesauri réalisés au coup par coup, pour remplir un rôle précis, puisque ils servent alors à couvrir un domaine que les outils classiques ne gèrent pas. Cette spécialisation ne dépend pas du fait que le thesaurus soit le résultat du travail d'experts (il s'agit alors de spécialistes d'une discipline, ou de documentalistes dans un domaine bien délimité), ou de l'extraction à partir d'ensembles documentaires, dont l'efficacité repose sur l'homogénéité lexicale des documents.

Lorsque ces thesauri sont utilisés dans le cadre de bases documentaires comportant plusieurs disciplines, voire des documents faisant appel au vocabulaire de plusieurs spécialités, l'utilisation de plusieurs de ces thesauri en parallèle devient inévitable. On peut alors décider de conserver plusieurs thesauri distincts, et les interroger tous pour chaque terme, ou de les réunir en un thesaurus unique.

La première méthode comporte cependant de nombreux inconvénients. Le coût algorithmique de la recherche de chaque terme est souvent logarithmique en fonction du nombre de termes contenus dans le thesaurus. Multiplier les recherches dans des listes de petites tailles entraîne donc un sur-coût algorithmique. En outre, multiplier les thesauri complique la gestion et l'administration des systèmes de TR associés.

Mais le principal obstacle est celui de la "normalisation" : deux thesauri dis-

tincts ont souvent des statistiques (nombre moyen de synonymes, connectivité) et une hiérarchie de termes (graphe d'hypéronymie) incompatibles. Dès lors, il est impossible d'accorder la même valeur aux résultats de leur interrogation pour un même terme si celui-ci apparaît dans les deux thesauri. Il n'est pas non plus possible de former, par exemple, un synset unique à l'aide des deux résultats obtenus : une simple réunion entraînerait un déséquilibre systématique en faveur de la spécialité associée au thesaurus pour lequel le nombre moyen de synonymes serait le plus élevé.

La seconde méthode, difficile, résout les différents problèmes de normalisation dès l'origine : des choix doivent être effectués, mais le thesaurus résultant est utilisable en l'état. Parmi ces choix se trouvent la question du rééquilibrage des thesauri (extension des synsets en incluant des termes à une distance supérieure à 1 sur le graphe de synonymie, restriction en éliminant certains synonymes sur des critères contextuels), et l'unification des hiérarchies. Dans les deux cas, il n'existe pas de méthode générique, et un travail humain important est de toute manière nécessaire.

La nécessité d'un rééquilibrage des thesauri provient du fait que les termes ou les sens particuliers des termes présents dans le thesaurus. pour lequel la taille moyenne des synsets est la plus élevée, seront favorisés. Prenons l'exemple d'une expansion sémantique de requête : chaque terme est remplacé par le synset associé dans le thesaurus utilisé. Sans rééquilibrage, la requête produite en utilisant un thesaurus issu d'une fusion contiendra en moyenne plus de termes issus du champ sémantique du thesaurus où les synsets sont les plus larges. Un moteur de recherche fournira alors plus difficilement des documents concernant le champ sémantique de l'autre thesaurus.

Pour réaliser ce rééquilibrage, le choix le plus élémentaire est celui de "l'élagage", qui consiste à réduire la taille moyenne des synsets du thesaurus où ils sont les plus larges. Comme il s'agit d'éliminer des termes, on va essentiellement se baser sur des critères statistiques. Un critère pourrait par exemple reposer sur le renforcement du lien de synonymie : si A est synonyme de B, B de C et C de A, alors A est "fortement" synonyme de B (et de C...), et le lien A-B sera conservé. Bien que cette méthode simple semble logique, elle ne donne pas forcément de bons résultats. Elaborer une méthode adaptée dépend fortement de la structure du thesaurus, et donc du "paysage" sémantique du domaine concerné. Ce qui implique encore une fois beaucoup de travail de la part de spécialistes, non seulement en sémantique, mais surtout d'experts du domaine.

La méthode opposée, qui consiste à étendre les synsets les plus petits, semble offrir des résultats plus fiables. La relation de synonymie est rarement transitive,

car dans ce cas il n'existerait que quelques sous-ensembles de synonymes dans un thesaurus. On met ce fait à profit, en introduisant de nouveaux termes obtenus par transitivité dans les synsets déjà existants : si A est synonyme de B et B de C, on considère que C devient aussi synonyme de A. Ce processus revient à considérer qu'un synset n'est plus l'ensemble des synonymes directs (les termes à une distance de 1 sur le graphe de synonymie) mais la classe plus large des termes à une distance inférieure ou égale à n. Le seul paramètre à régler est alors cet entier n, ce qui permet un "réglage" rapide, et limite l'intervention humaine. Dès lors que ce paramètre a été identifié, le thesaurus étendu peut être reconstruit automatiquement.

Thesauri flous

Comme on l'a évoqué plus haut en parlant d'extension de synsets, un thesaurus dispose d'une métrique intrinsèque : la distance qui sépare deux termes est la longueur du plus petit chemin de synonymes les reliant. Cette distance ne peut donc prendre que des valeurs entières. Aussi les transformations se basant sur cette mesure de distance sont-elles grossières, comme l'expansion décrite plus haut, qui étend généralement beaucoup les synsets, même quand n est fixé à 2.

La notion d'une "gradation" d'intensité dans la synonymie est relativement évidente : on conçoit bien que si "long", "grand" et "gros" servent tous trois à désigner une taille importante, et sont à ce titre synonymes, les deux premiers sont plus facilement interchangeables.

C'est sur cette notion que s'appuie une version modifiée des thesauri classiques, dans laquelle chaque relation de synonymie est accompagnée d'un poids, représentant sur une échelle arbitraire la proximité de ces deux termes. Pour éviter toute confusion, dans ce document, nous désignerons ces structures par le terme *thesaurus flou* (voir [Bra77] pour une analyse du concept, et [Bro77] pour un cas d'application à l'extraction d'information). Grâce à ces informations, la distance associée au thesaurus, à valeurs réelles et non plus entières, donne alors tout son sens à l'arsenal des notions et outils de la topologie métrique : boules, frontière et densité.

On dispose alors d'une extension de la notion de synset. Dans le modèle classique, le synset associé à un terme était la boule de rayon 1 centrée sur ce terme. On définit maintenant le δ-synset comme la boule de rayon δ centrée sur le terme. Lorsque l'on réalise une expansion sémantique grâce à ce type de thesaurus, le paramètre δ permet de régler la largeur de l'étendue sémantique concerné par l'expansion.

S'il était possible dans le cadre des thesauri classiques de définir la distance de deux ensembles de termes (elle était soit nulle, soit supérieure à 1), sa valeur n'avait alors pas de traduction sur le plan sémantique. Au contraire, la notion de distance d'ensembles de termes dans le cadre des thesauri flous acquiert une réelle signification, en tant que mesure de l'écartement des domaines de sens que recouvrent ces deux ensembles. Cet aspect est particulièrement important si l'on dispose, pour chaque document, d'un ensemble de mots-clés, déterminés par des experts ou extraits automatiquement (voir section 2.4). On dispose alors immédiatement d'une mesure sémantique de la distance de deux documents.

Cependant, la fiabilité de tous ces instruments dépend de l'homogénéité de la mesure de synonymie. Qu'une partie du thesaurus ait de manière systématique des distances de synonymie plus faibles qu'une autre serait désastreux, par exemple, pour établir la distance de deux documents ayant chacun des mots clés dans ce domaine **et** dans d'autres : la valeur de distance (qui est la plus petite des distances entre termes pris deux à deux) serait alors systématiquement une distance entre deux termes de la zone où les distances sont les plus petites. Ce qui revient à systématiquement surestimer l'importance des termes appartenant à cette zone.

Cette mesure de synonymie relève d'une échelle arbitraire, puisqu'elle dépend uniquement du sens des termes, ce qui interdit toute mesure absolue. Et garantir la cohérence d'une telle échelle lorsque les valeurs en sont données par des êtres humains, documentalistes ou experts, est très difficile. Il est possible de s'en approcher lorsque les évaluations sont obtenues comme une moyenne d'un groupe important de personnes, les variations individuelles étant alors atténuées, mais ceci a un coût financier important.

En revanche, assurer cette cohérence lorsqu'on emploie des méthodes d'extraction automatiques à partir d'un corpus documentaire est beaucoup plus aisé. Ou, plus exactement, le biais enregistré dans le thesaurus sera celui du corpus documentaire lui-même. Constituer un ensemble de documents sans biais notable lorsque l'on dispose de bases documentaires correctement indexées est relativement peu coûteux. Si l'on emploie le thesaurus extrait pour faire de l'expansion de requêtes, un biais eventuel reflètera celui du corpus documentiare, et ne sera donc pas pénalisant dans les résulats de requêtes obtenus.

Les différentes méthodes d'extraction décrites sont en pratiques utilisées en fonction du type et de la richesse des informations présentes dans la base, d'une part, et du type de thésaurus que l'on souhaite obtenir (classique, ou flou). La plus grande partie de ces méthodes fournit une évaluation numérique de l'intensité des relations de synonymie. Celle-ci est plus ou moins proche d'une évalua-

tion humaine selon les méthodes, mais possède l'avantage d'être reproductible, puisqu'obtenue par un algorithme de calcul.

2.4 LSI

Parmi les différentes méthodes d'extraction sémantique utilisables pour la construction de thesauri, la méthode LSI (ou *Latent Semantic Indexing*, formalisée par S. Deerwester dans [DeeDum+88] puis [DeeDum+90]), a prouvé son efficacité dans de nombreuses situations. Surtout, il s'agit d'une méthode numérique statistique, applicable à toutes les langues, sans présupposé de structure grammaticale ou d'énonciation[4]. Il est en outre possible de contrôler efficacement son coût algorithmique.

Latent Semantic Indexing

S. Deerwester a proposé dans [DeeDum+90] deux techniques d'analyse sémantiques s'appuyant sur des techniques de réduction de dimensionnalité.

La version unimodale repose sur une analyse en composantes principales d'une matrice homogène carrée de degrés d'association entre documents. Chacun de ces degrés représente une mesure empirique de similarité entre document, par exemple une estimation humaine de similitude, ou une mesure de recouvrement lexical. La diagonalisation de cette matrice (symétrique) permet d'identifier les vecteurs propres associés aux plus grandes valeurs propres, porteurs de l'essentiel de la mesure d'association. Il est ainsi possible d'exprimer le contenu d'un document, du point de vue de la mesure d'association adoptée, par quelques valeurs numériques.

La version multimodale utilise une matrice rectangulaire relevant les occurences de chaque terme dans chacun des documents. Une décomposition en valeurs singulières permet là encore de ne conserver que les composantes de plus grand poids. Mais contrairement à la première méthode, celle-ci fournit une expression réduite pour chaque document ainsi que pour chaque terme. Elle peut donc servir de base à une analyse sémantique, et c'est celle qui nous intéressera dans la suite.

[4] Ceci permet, en particulier, son utilisation pour l'analyse des langues asiatiques, pour lesquelles les relations causales ne sont pas, à l'inverse des langues indo-européennes, l'élément structurant de l'énonciation.

LSI : approche mathématique

Considérons un ensemble documentaire. Il est possible, par une des méthodes d'extraction de termes citées plus haut, d'obtenir la liste $\mathcal{W}$ des termes de ces documents (rappelons qu'il s'agit de la forme lexicale, forme unique des mots signifiants présents dans les documents).

Représentons par $\mathcal{D}$ l'ensemble des documents de la base. Relever l'occurence de chaque terme dans chaque document fournit une matrice rectangulaire $\mathbf{M}$, dont l'élément $M_{w,d}, w \in \mathcal{W}, d \in \mathcal{D}$ est le nombre d'apparitions du mot w dans le document d.

Il existe des matrices $\mathbf{U}$, $\mathbf{V}$ dont les colonnes sont orthonormales, et $\mathbf{D}$ diagonale de valeurs diagonales décroissantes, telles que $\mathbf{M} = \mathbf{U} \times \mathbf{D} \times \mathbf{V}^t$.

Fixons un entier p, et notons $\mathbf{D}_p$ la matrice diagonales ne comportant que les p plus grandes valeurs diagonales de $\mathbf{D}$. La matrice de rang p la plus proche (par la distance quadratique) de $\mathbf{M}$ est $\overline{\mathbf{M}}_k = \mathbf{U} \times \mathbf{D}_k \times \mathbf{V}^t$.

Celle-ci nous donne accès à une mesure simplifiée de la similitude entre termes (produit scalaires de lignes de $\overline{\mathbf{M}}_k$) et entre documents (produit scalaires de colonnes de $\overline{\mathbf{M}}_k$). Enfin, une mesure de la relation "latente" (i.e. non exprimée) entre un terme et un document est donnée directement par la valeur $\overline{\mathbf{M}}_{k\,w,d}$.

L'intérêt de ces nouvelles mesures (discuté dans [DeeDum+90]) est de s'abstraire de la nécessité de présence explicite de termes dans les documents pour produire une trace dans la mesure de similarité, lorsque l'ensemble du contexte soutient la notion véhiculée par ce terme.

3 Personnalisation, réactivité et data-repositories

Comme on l'a évoqué plus haut, l'accès aux bases de données documentaires se fait principalement au travers d'un intranet ou de l'internet. Ces bases sont stockées sur des installations dédiés, et les utilisateurs les consultent au moyen d'un navigateur internet, via le protocole HTTP. Ce mode d'accès distribué impose un modèle d'interaction appelé client-serveur, dans lequel l'utilisateur – le client – envoie des demandes au serveur qui possède la base consultée. Son caractère asynchrone permet de nombreux accès concurrents et une grande flexibilité dans la compatibilité des matériels.r

Il pose cependant des contraintes notables tant sur le plan des IHM que des capacités techniques. Il est en fait source d'une grande partie des c ontraintes

pesant sur le type de fonctionnalités qu'il est possible de proposer à l'utilisateur dans les outils de TR accessible par HTTP. Et il limite étroitement les informations qu'il sera possible d'acquérir auprès de l'utilisateur. Il nous faut donc en inventorier précisément les capacités affin d'en tirer partie au mieux.

3.1 Client-side

Le protocole HTTP lui même ne restreint pas les modes d'interaction accessibles à l'utilisateur. Cependant, il est le plus souvent associé au HTML, format de visualisation des documents permettant, depuis la version 4.0, de créer des interfaces réactives. Pour permettre la portabilité de ce format sur tout type de système et de matériel, les interactions possibles entre l'utilisateur et l'interface sont précisément codifiées, à travers les spécifications publiées par le **W3C** [@W3C]. Nous nous référerons ici à la RFC 2616 [HTTP-1.1], qui précise les caractéristiques de HTTP 1.1, et à la RFC 1866 [HTML-2.0], qui spécifie le format HTML 2.0. Le lecteur peut trouver dans la spécification HTML 4.01 du W3C [HTML-4.0] un complément de lecture intéressant concernant les aspects dynamiques de HTML, dont nous ne faisons pas usage ici.

Interaction web

Le lecteur a probablement l'expérience des modes d'interaction sur le web : suivi de liens, base de la navigation classique, listes déroulantes, ou menus, pour ce qui est des actions de l'utilisateur, texte et images, parfois animés, pour ce qui est de la présentation. Il s'agit donc le plus souvent de pages "passives", se comportant comme les pages d'un ouvrage papier tant que l'on ne clique pas sur un lien ou que l'on n'active pas une sélection. La réactivité est limitée à déclencher une action et à attendre la réponse, ce qui revient en quelque sorte à "tourner la page".

En particulier, les types d'interaction que l'on attend d'un logiciel classique – réponse immédiate à l'action du clavier, utilisation dynamique de la souris, comme aide contextuelle, drag-and-drop, sélection multiple – ne sont généralement pas disponibles[5].

Cette interaction limitée résulte essentiellement de la nécessité d'interroger un serveur distant pour connaître la réponse à apporter à chaque action. Ce qui implique des délais de connection et de traitement du côté serveur, et des

[5] De fait, HTML 4.0 était destiné à répondre à cette carence, mais la mise en œuvre de tels modes d'interaction reste très complexe.

délais d'analyse et d'affichage du coté client. En effet, pour diminuer la quantité d'information qu'il est nécessaire de faire transiter sur les réseaux, les données envoyées au client sont limitées au strict nécessaire pour l'affichage. Le client ne dispose donc pas des éléments de décision permettant d'élaborer lui-même la réponse aux actions de l'utilisateur. Celle-ci doit être demandée au serveur, qui dispose de ces données.

Dans une page HTML, une interaction peut être réalisée soit par l'accès à un lien (adresse immuable d'un autre document) présent dans la page, soit par le biais de formulaires, permettant une saisie textuelle et le passage des textes saisis à un gestionnaire sur le serveur. Dans le cas d'un lien, celui-ci contient toutes les informations permettant au serveur de trouver ou générer la page correspondante. Dans le cas de formulaires, celui-ci doit d'abord analyser les informations fournies par l'utilisateur. Si, comme nous le verrons plus loin, il existe en fait peu de différence du point de vue du serveur, les types d'interaction accessibles par ce moyen sont très différents : sélectionner un lien dans une liste est une action rapide, fournir des information pour accéder à un document est beaucoup plus lent. La liste de liens est donc le mode d'interaction privilégié, en particulier s'il s'agit de faire une sélection au sein d'un grand nombre de choix.

Lorsque cette interaction a été réalisée, les informations envoyées au serveur se présentent dans les deux cas de la même manière : une chaîne de requête. Celle-ci comporte une adresse identifiant le document recherché, et éventuellement les données entrées par l'utilisateur. Cette adresse unique, ou URI (Unified Resource Identifier), et dépendante du serveur, permet à celui-ci de retrouver le document demandé ou le gestionnaire devant traiter cette requête. Outre les données fournies par l'utilisateur, le serveur a aussi accès à un identifieur du type de client utilisé, à l'adresse de la machine d'où provient la requête, et à l'instant d'envoi de la requête.

3.2 Server-side

Du point de vue du serveur, les possibilité de traitement de l'information sont théoriquement illimitées. En pratique, les contraintes proviennent de la quantité d'information réduite associée aux requêtes HTTP, des bandes passantes des réseaux réduisant la quantité d'information que l'on peut raisonnablement envoyer aux clients, et des capacités réelles de traitement du serveur. Ces limitations de capacité proviennent du coût élevé des matériels adaptés et des licences des logiciels permettant de procéder à des traitements élaborés de l'information.

Rappelons, pour donner un aperçu de ces contraintes, quelques ordres de

grandeur. Considérons le contexte d'un intranet d'entreprise : un document de travail typique, comportant une centaine de pages de texte et quelques images, peut aisément atteindre 500ko. Si le serveur de l'entreprise reçoit de l'ordre de 10000 demandes de pages par journée de 8h, (il s'agit d'un nombre relativement faible, pour une entreprise comptant un effectif d'une cinquantaine d'utilisateurs de l'outil informatique), cela correspond à un débit de données moyen de 1,4 Mb/s. Le débit maximal des réseaux – physiques – de bien des entreprises est de 10 Mb/s. Bien évidemment, le débit réel – et donc la charge de travail pour le serveur – est beaucoup plus irrégulier, et on atteint très vite les limites physiques des réseaux. Un serveur permettant de traiter de tels volumes, avec des temps de réponse acceptables, coûte entre 50 et 100 fois le prix d'un PC bureautique. C'est aussi le coût moyen des licences de solutions intégrées de gestion des données, de la base documentaire aux gestionnaires d'intranet.

Les contraintes techniques sont donc fortes, les deux principales étant de conserver des volumes de données transférées faibles, et une utilisation CPU basse pour les gestionnaires et outils d'analyse de données.

Serveurs de données

Les contraintes conditionnant les caractéristiques des serveurs intra- et internet concernent les trois éléments de la chaîne de service des données : le stockage (disques durs et archivage), le traitement (gestion des requêtes), et le transport (réseaux de donnée, ou ethernet). Lorsque les quantités de donnés concernées ou le nombre de demandes sont importants, ces trois fonctions peuvent être attribuées à des matériels distincts.

Les pages et documents fournis aux utilisateurs sont soit "statiques", ne nécessitant pas de transformation avant d'être fournies aux clients, soit "dynamiques", construites à la volée (au moment de la requête) en fonction des données fournies par le client, et de données brutes stockées sur le serveur. On se trouve toujours dans le cas dynamique lorsque la requête est issue d'un formulaire, pour permettre le traitement des données fournies par l'utilisateur. Les pages statiques font surtout appel au stockage et au transport, le stade de traitement se limitant à faire la relation entre adresse demandée et document stocké.

Il est en premier lieu nécessaire de construire la base de données documentaire, et de la rendre accessible. Cette étape, qui sort de l'objet de cette recherche, demande des dispositions particulières : redondance de supports de stockage, systèmes d'archivage, matériels permettant une maintenance en cours de fonctionnement. Leur accessibilité est assurée grâce à des systèmes adaptés à de gros

débits de donnés. Dans le cas de pages dynamiques, le volume de données devant être "manipulées" est en effet généralement bien supérieur au volume final du document généré (d'un facteur pouvant atteindre 1000 dans le cas de bases de données). Cette asymétrie permet de limiter le volume de données à faire transiter sur les réseaux.

L'étape de transport nécessite la mise en oeuvre de switchs rapides (matériels permettant l'interconnection de réseaux), et lorsque la disponibilité est un élément crucial, de mécanisme de répartition du trafic généré par les requêtes sur plusieurs serveurs. Cette technique permet en outre de diminuer l'impact des opérations de maintenance, en n'arrêtant qu'un serveur à la fois.

Cependant, c'est sur l'étape de traitement, lorsque celle-ci est extensive, que porte le plus grand nombre de contraintes. D'une part, elle doit faire face aux limitations des deux autres phases – en limitant les débits de données en entrée ou en sortie – mais c'est le plus souvent elle qui amène le plus rapidement les matériels à leurs limites : mémoire et capacité de calcul. Le traitement de l'information reste le plus gros goulet d'étranglement des systèmes de TR actuels.

Lorsqu'il s'agit de pages dynamiques, plusieurs solutions de traitement sont d'une utilisation répandue. Parmi les plus courantes, citons PHP, permettant d'intégrer dans une page HTML les résultats de requêtes simples dans une base de données, ASP et JSP. Mais toutes répondent à un format d'interaction construit au-dessus de HTTP, appelé CGI, ou Common Gateway Interface.

3.3 CGI et traitement de l'information

L'internet (ou sa version localisée, intranet) est le moyen de choix d'accès aux informations électroniques. Les protocoles utilisés fixent le cadre dans lequel il est possible de fournir des informations spécifiques à la demande d'un utilisateur, et le moyen d'assurer le suivi et la pérennité dune telle transaction.

Depuis sa création, internet permet de proposer à la consultation information et documents. Stockée sous forme numérique, cette information (texte, images, sons, et autres formes de données) est mise à jour soit manuellement, soit automatiquement, mais de manière indépendante de toute demande de l'internaute. La notion de "service internet", d'apparition récente, recouvre non seulement l'accès à des données, mais surtout à une version mise en forme, pré-traitée, de ces données stockées. Il s'agit de pages dites "dynamiques", répondant à une requête construite à partir des informations fournies par l'utilisateur. L'acquisition de ces données repose sur l'utilisation de formulaires, permettant l'entrée d'informations dans une page HTML. Ces informations sont envoyées suivant le

protocole CGI. Elles sont traitées sur le serveur par ce que l'on appelle communément un script CGI (petit programme spécialisé dans cette tâche), qui produit en retour un document (généralement une page HTML, mais il peut aussi s'agir d'une image ou de tout autre type de document) contenant les informations demandées par l'utilisateur.

Le protocole CGI [@CGI] se contente en fait de spécifier le format des donnés complémentaires envoyées par l'utilisateur. En effet, outre l'adresse du document demandé, le protocole HTTP spécifie que des donnés complémentaires peuvent être envoyées par le client – il s'agit généralement des donnés saisies dans un formulaire – selon deux méthodes, GET et POST. L'utilisation de CGI impose à ces données de prendre la forme de couples **variable**=*valeur*, la valeur étant une chaîne de caractères ASCII[6]. Le codage des donnés d'un formulaire – où chaque zone d'entrée de données possède un nom – par les clients doit suivre ce modèle, en affectant les donnés d'un champ nommé à la variable du même nom.

Soulignons que la création de pages dynamiques via CGI (qu'il s'agisse de PHP, ou d'autres langages de script) fait reposer l'essentiel de la charge sur le serveur. Une approche orthogonale consiste à déléguer la part la plus importante des traitements à effectuer au client. C'est par exemple le cas pour des applications en ligne à base de Java (langage de programmation d'applications multi-plateforme) et pour les pages dynamiques à base de Javascript et de HTML 4.0 (ou DynHTML). Dans les deux cas, le client reçoit un code plus complexe, incluant les aspects de réactivité et d'interactivité. Rien n'interdit cependant d'utiliser ces deux méthodes conjointement, c'est d'ailleurs souvent le cas dans certaines applications internet, comme les services aux internautes (mail et hébergement), le commerce électronique ou encore les services financiers. On retrouve encore cette méthode dans les outils de recherche les plus évolués, destinés aux intranets d'entreprise (dont une mise en oeuvre est décrite dans [VacPar+01]).

3.4 Client-side ou server-side ?

Dans de nombreux cas (qu'il s'agisse de moteurs de recherche, de commerce en ligne, ou de services de messagerie[7]), la notion de service sur le web consiste à élaborer interactivement une présentation (ou "vue") adaptée d'un ensemble de données relativement important : ensemble des documents indexés, catalogue

[6] L'utilisation de caractères non-ASCII peut se faire en utilisant un codage numérique, ou %num représente le caractère de code **num**

[7] Par exemple : Altavista et Google, déjà cités, E-Bay et Amazon, Yahoo mail et MSN, et bien d'autres applications...

de produits, facture, boite aux lettres électronique,... Un éditeur de services peut concevoir deux approches permettant d'obtenir ce résultat : fournir à l'utilisateur les données et un logiciel approprié pour les traiter, ou réaliser lui-même le traitement, et ne fournir que les vues demandées. Outre la question des coûts respectifs de ces solutions et de la confidentialité des données, une des premières considérations intervenant dans ce choix est celle de la bande passante.

Dans le cas d'un moteur de recherche par exemple, la taille des données nécessaires se compte en téra-octets, et nécessite des équipements spéciaux, comme on l'a évoqué plus haut. Il n'est donc pas envisageable de traiter ces données au niveau du client, d'autant plus que la part de ces données réellement utilisée (effectivement fournie en réponse aux requêtes de l'utilisateur) est infime. La centralisation des données et de leur traitement facilite la mise à jour des contenus, leur maintenance et leur protection (contre les destructions accidentelles ou criminelles, l'abus, et les accès non autorisés). Elle assure aussi l'accessibilité du service, dans la mesure où le seul logiciel nécessaire pour les clients du service est un navigateur web, comprenant HTTP et HTML, outil universellement répandu.

En revanche, lorsque le volume de données est limité, un logiciel dédié permet de s'affranchir des limitations de HTTP et HTML. Se pose alors la question de la sécurité des donnés transférés, des mises à jour du logiciel, de la vérification de son innocuité, et des plates-formes sur lesquelles on pourra l'utiliser. Le langage Java (développé par **Sun Microsystems**, voir [@JAVA]) tente de répondre à ces deux dernières préoccupations, à l'aide d'une machine virtuelle offrant un environnement d'exécution des logiciels standardisé et sécurisé. Java est aujourd'hui utilisé dans de nombreux systèmes interactifs, et fournit de nombreux outils permettant en outre de sécuriser les données. Il introduit cependant de nouvelles limitations (en termes de performance et d'interaction avec le matériel), et ne peut apporter de solution au problème des mises à jour logicielles.

State-less et state-full

Lors de services réalisés au moyen de pages dynamiques, l'obtention du résultat final souhaité par l'utilisateur est souvent obtenu après plusieurs étapes de sélection, faisant appel à la présentation par le serveur de résultat intermédiaires. Le résultat final est ainsi "négocié" au long de plusieurs cycles requête-réponse. Il est donc nécessaire de conserver, sur plusieurs requêtes HTTP, les informations sur l'état d'avancement de cette négociation, pour un utilisateur particulier.

Cependant, HTTP, élaboré dans l'optique de l'accès à des documents sta-

tiques, n'offre aucun moyen de conserver cette information sur plusieurs requêtes. Il s'agit d'un protocole "state-less", pour lequel chaque requête est unique et isolée. La diversité des notions d'identifiant utilisateur, et l'existence des connections dites "dial-up" qui peuvent, à l'issue d'une reconnection, entraîner un changement de l'identifiant de la machine cliente (adresse IP), rendraient de toute manière difficile l'implémentation d'un protocole fournissant cette possibilité. En outre, un utilisateur peut mener de front plusieurs négociations de ce type, qu'il faudrait savoir distinguer.

Les solutions à cette limitation très gênantes passent par la notion de "session" : un ensemble de plusieurs requêtes connectées. La première requête d'une session doit être une ouverture, permettant de générer les informations nécessaires. En l'absence de standard, c'est le serveur qui en impose le format, le client se contentant de rappeler ces informations à chaque requête ou lorsqu'on le lui demande.

Deux mécanismes utilisent cette méthode : les "cookies", bloc d'informations fourni par le serveur au client à l'initiation de la session, et rappelé par le client lors de ses requêtes suivantes, et CGI, qui stocke ces données parmi les variables utilisateur, et les rappelle dans les réponses aux requêtes. On obtient dans les deux cas une connection "state-full", encore appelée "state-aware", permettant l'identification d'un échange avec un utilisateur particulier, et le suivi de sa progression.

3.5 Identification et personnalisation

Si le rappel d'un bloc de données caractéristique d'une session à chaque requête HTTP permet d'obtenir une connection "state-full", en revanche un tel système n'apporte pas de manière intrinsèque une méthode fiable d'authentification des utilisateurs, ni d'assurance que plusieurs sessions peuvent être gérées en parallèle par un même client.

Authentifications et données personnelles

L'authentification d'un utilisateur (déclaration et vérification de son identité) passe généralement par un couple identifiant / mot de passe. Si l'identifiant est une donnée publique, le mot de passe est un élément précieux, qui ne permet une vérification d'identité que s'il n'est connu que de l'utilisateur et du serveur qui doit le vérifier. Pour limiter les risques, il n'est le plus souvent utilisé qu'une fois, lors de l'établissement d'une session.

Le mécanisme de suivi de session utilisateur le plus utilisé repose sur la fabrication, lors de l'ouverture de session, d'un "jeton" caractéristique de la session, à validité limitée. Le serveur maintient une table des sessions ouvertes, identifiées par leur jeton, unique, et permettant grâce à celui-ci de retrouver les informations associées : utilisateur, état courant de la session, date d'ouverture, de la dernière requête, adresse du client... Dans certaines situations, le serveur peut juger le jeton invalide, et clore d'office la session en supprimant la session : temps d'inactivité trop long, adresse différente, opération interdite...

C'est ce jeton, fourni au client lors de l'ouverture de session, qui est rappelé à chaque requête (via un cookie ou une variable CGI), et non les données "sensibles" que sont l'identifiant et surtout le mot de passe. Couplé à une base de données des utilisateurs, permettant de vérifier et maintenir ces éléments, ce mécanisme permet une authentification fiable des utilisateurs, et la gestion de plusieurs sessions d'un même utilisateur (le jeton étant caractéristique d'une session, et non d'un utilisateur).

La base de données utilisateur peut contenir aussi de nombreuses informations permettant d'améliorer le traitement des requêtes : options, préférences réglées par l'utilisateur une fois pour toutes, données personnelles (sélections favorites, adresse d'envoi des achats, contact téléphonique, etc). Chaque requête entraînant une vérification du jeton, il est en outre aisé de produire un historique des actions effectuées, permettant à l'utilisateur de répéter ses dernières actions, et au créateur du service d'extraire des analyses statistiques de ces données permettant d'en améliorer l'efficacité.

On atteint alors des niveaux de confidentialité, de sécurité et d'interactivité dignes d'un logiciel "classique", permettant d'appliquer la plupart des techniques du développement logiciel en matière d'ergonomie et d'IHM.

Temporalité et client-serveur

Lorsqu'on utilise ces outils, il reste cependant un élément dont on ne peut s'affranchir : il existe un certain délai, incompressible, entre une action au niveau de l'interface HTML, qui se traduit par une requête, et le retour de la réponse générée par le serveur. Ce délai est dû au transit de ces informations sur le réseau internet, composé de sous-réseaux interconnectés par des routeurs, qui introduisent chacun un délai faible, mais répété. Si, dans une interface classique, une action se traduit immédiatement par une réaction, le délai systématique peut entraîner gêne et fatigue chez les utilisateurs d'interfaces web.

Ce délai est particulièrement gênant lorsqu'il est nécessaire de mesurer le

temps de réaction d'un utilisateur. En effet, on ne peut effectuer cette mesure sur le serveur, qu'entre deux requêtes lui parvenant. Il n'est alors pas possible de prendre en compte le délai de transfert, très variable d'une requête à l'autre. Toute mesure du temps dans le cadre d'une interface web, qu'il s'agisse de temps de réaction ou de la durée passée à effectuer une tâche (comme le temps de consultation d'un document, le temps d'inactivité de l'interface, ...) doit donc être traitée avec la plus grande précaution.

4 Algorithmes évolutionnaires

Nous envisagerons ici les Algorithmes Evolutionnaires sous l'angle des méthodes d'optimisation. A cette fin, il est utile de préciser le formalisme couramment utilisé. Dans ce cadre, on se fixe une fonction dite *objectif*, à valeurs numériques, définie sur un espace d'états muni d'une topologie. Nous appellerons ce dernier *espace de recherche*. Réaliser une tâche d'optimisation sur cette fonction objectif $f : \mathcal{E} \mapsto \mathbb{R}$ consiste à trouver l'ensemble $X = \mathrm{a}rgmax_{x \in \mathcal{E}} h(x)$ des éléments de l'espace de recherche maximisant la fonction objectif.

4.1 Le paradigme évolutionnaire

Les Algorithmes Evolutionnaires, paradigme d'optimisation fondé sur le modèle darwinien de l'évolution naturelle, offre une alternative aux méthodes traditionnelles :

- déterministes, imposant une connaissance approfondie des propriétés mathématiques de la fonction objectif,
- stochastiques, permettant difficilement d'améliorer la performance des algorithmes en introduisant une connaissance *a-priori* du problème.

Inspiration et principe

En simplifiant à l'extrême, le modèle de l'évolution darwinienne postule que le génotype des individus, qui s'exprime de manière visible dans le phénotype, évolue de génération en génération grâce à la reproduction et aux mutations, sous la pression sélective de l'environnement naturel.

Le paradigme évolutionnaire s'inspirent de ce principe, en remplaçant la notion de pression environnementale par une évaluation numérique de l'individu, et le génotype par une série de paramètres définissant les caractéristiques d'un individu. L'implémentation de ce paradigme requiert que l'on se fixe :

- un codage des individus, définition précise des paramètres suffisant à décrire ceux-ci,
- une fonction d'évaluation, algorithme permettant de calculer une valeur numérique de la performance de l'individu,
- une méthode d'initialisation, permettant de créer de nouveaux individus "au hasard",
- une méthode de sélection, permettant de choisir quels individus seront amenés à se reproduire,
- une méthode de reproduction, algorithme permettant de produire un nouvel individu en utilisant les caractéristiques d'un ou plusieurs individus existants,
- une taille critique, éventuellement variable, indiquant la taille d'échantillonnage nécessaire au fonctionnement de l'algorithme,
- un critère d'arrêt, permettant de décider que l'on a obtenu la (les) solutions recherchée(s).

Après la validation du critère d'arrêt, la solution est obtenue comme l'individu de la dernière population possédant la meilleure évaluation. Il s'agit bien entendu d'une approximation numérique, et ne correspond à une solution du problème d'optimisation associé à la fonction d'évaluation que dans la mesure où celui-ci possède une solution unique et bien définie.

L'algorithme évolutionnaire consiste alors à successivement :

1 initialiser la population, pour obtenir un nombre d'individus égal à la taille critique (il s'agit de la première génération),
2 évaluer tous les individus,
3 appliquer la méthode de sélection pour produire un p-uple d'individus, à partir duquel est effectuée la reproduction, fournissant ainsi un nouvel individu,
4 répéter l'étape 3 jusqu'à obtenir assez d'individus pour constituer une population de taille adéquate (la génération suivante),
5 revenir à l'étape 2 jusqu'à la validation du critère d'arrêt.

Les premières implémentations ont été réalisées suivant un modèle appelé algorithme génétique (AG), encore employé dans la plupart des applications des AE. Les AGs utilisent une chaîne de bits pour génome, et deux opérateurs de reproduction : le crossover, produisant une nouvelle chaîne par échange de sections déterminées stochastiquement entre deux chaînes parentes, et la mutation, inversant certains bits aléatoirement. Les méthodes de sélection reposent sur une comparaison des évaluations des individus. Trois méthodes de sélection sont utilisées très couramment :

- La sélection en tournoi tire des paires d'individus au hasard parmi les n

premiers, et renvoie les individus possédant l'évaluation la plus élevée dans leur paire.
- La sélection par rang classe les individus par évaluation croissante, et tire des individus avec une probabilité proportionnelle à leur rang.
- La sélection en roulette tire les individus avec une probabilité proportionnelle à leur évaluation.

Il existe cependant de nombreuses autres méthodes, développées pour répondre à des problématiques précises.

Du point de vue de l'algorithme d'óptimisation, la relation entre évaluation et fonction objectif n'est pas toujours immédiate. En effet, il est rare que le codage adopté soit une expression directe des élément de l'espace de recherche. On retrouve ici la distinction entre phénotype (élément de l'espace de recherche) et génotype (codage associé). Elle permet de choisir un codage adapté à la définition d'outils de reproduction algorithmiquement efficaces et peu coûteux, mais introduit la nécessité d'exprimer la fonction objectif en fonction des génotypes. De plus, il est souvent possible et souhaitable d'orienter l'algorithme, en introduisant une correction dans l'évaluation pour, par exemple, éliminer les individus qui ne respectent pas certaines contraintes pratiques. Dans les cas les plus simples, l'évaluation est directement la valeur de la fonction objectif du phénotype associé.

Vocabulaire et théories

Nous avons introduit dans le paragraphe précédent de nombreux termes spécifiques de la théorie évolutionnaire, de manière informelle. Nous allons tenter ici d'en donner une définition précise. Bien que les méthodes évolutionnaires aient été appliquées sous des formes variées, ces définitions restent valables dans la plupart de ces contextes. La description de l'algorithme associé que nous donnons figure 2.5 décrit elle aussi une grande partie des algorithmes évolutionnaires utilisés couramment.

Individu : Un point d'échantillonnage de l'espace de codage (ou espace des génomes).

Fitness : Evaluation de performance d'un individu (proximité de l'individu à un optimum, par exemple). Celle-ci évalue les génomes, et est donc basée sur la composition d'une transformation, amenant de l'espace de codage à l'espace de recherche, et de la fonction objectif.

Population : Echantillonnage de l'espace de codage, par une collection d'individus.

Opérateur génétique : Tout algorithme permettant de créer un nouvel individu à partir d'un p-uple d'individus, qu'il soit stochastique ou déterministe.

Reproduction : Production, grâce au mécanisme de sélection et à l'application des opérateurs génétiques, d'un nouvel individu à partir d'un p-uple d'individus pris dans la population courante.

Génération : Une itération de l'algorithme d'optimisation. A chaque génération est associée une population, et l'on passe d'une génération à la suivante en constituant une nouvelle population par reproduction des individus de la population courante.

Sélection : Mécanisme fournissant un p-uple d'individus à partir d'un ensemble d'individus et de leurs évaluations.

Mutation : Opérateur génétique unaire introduisant des modifications aléatoires dans le génome d'un individu (usuellement de faible amplitude).

Crossover : Opérateur génétique binaire (dans la plupart des cas), produisant un nouvel individu par recombinaison de parties des génomes qui lui sont fournis.

Notons que la distinction qui est faite ici entre génome et individu n'est que rarement formalisée. Elle n'est nécessaire que dans le cas où certaines formes de génomes, permises par le format de codage adopté, n'ont pas d'équivalent dans l'espace de recherche.

Ces différents mécanismes participent, chacun à leur tour, à la création d'une nouvelle population à partir de la population courante évaluée, comme le montre le schéma 2.5.

Comparativement à d'autres outils d'optimisation stochastique, comme le recuit simulé ou les méthodes de Monte-Carlo, il existe peu de modèles mathématiques amenant à des prédictions de convergence pour les algorithmes évolutionnaires. Cela tient à la généralité du formalisme adopté : il n'existe pas de contrainte *a-priori* sur le codage adopté, les opérateurs génétiques peuvent êtres extrêmement variés, et aucune hypothèse n'est faite sur la fonction de fitness. Aussi la plupart des résultats concernent un type particulier – le plus courant – d'algorithme évolutionnaire : l'algorithme génétique à chaîne binaire. Même dans ce cas, sans spécification supplémentaire de la fitness, les résultats théoriques obtenus ne portent que sur la convergence, sans donner d'éléments de performance : vitesse et précision de la convergence (voir par exemple [Aga97, TDav91]). Toute théorie asymptotique appelle des hypothèses restrictives sur le codage et les opérateurs utilisés. Des résultats précis, mais d'utilisation toutefois délicate, sont obtenus pour des AG binaires, comme dans [Rud97, Fra98, Cer95]. Dans le cas

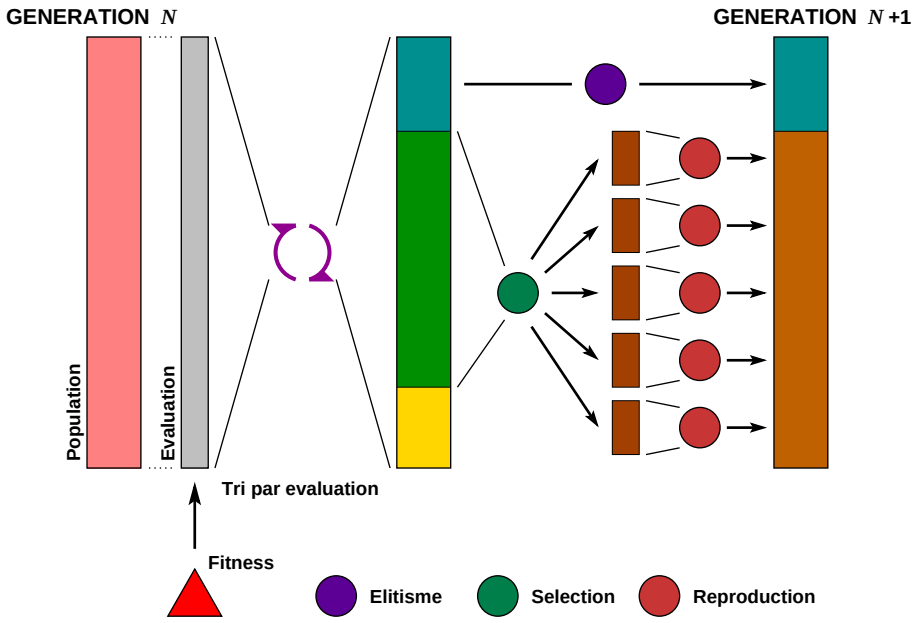


FIG. 2.5 – Diagramme d'évolution des populations d'un algorithme évolutionnaire.

général, il est possible d'obtenir à partir d'hypothèses du même type des résultats de vitesse de convergence, mais l'évaluation numérique de ces vitesses reste très difficile (voir [YLSLut00]).

4.2 L'approche parisienne

Des travaux théoriques, autant que l'observation pratique des populations finales obtenues dans des cas pratiques d'application des EA, révèlent que l'information présente dans ces populations ne se limite pas à la position de l'optimum global. La position des optima locaux, la difficulté de l'approche de l'optimum, les bassins d'attraction associés, et parfois des caractéristiques d'intérêt spécifique au problème posé, sont autant d'informations présentes dans les populations.

Basés sur une généralisation des systèmes de classeurs [Gol89], l'approche parisienne tente d'exploiter au mieux cette capacité, en n'examinant plus seulement le meilleur individu de la population en tant que candidat à une solution unique, mais l'ensemble de la population comme une solution globale. De manière similaire aux approches du type co-évolution, il s'agit de construire une solution "partagée", en faisant collaborer les individus. Cela introduit bien sûr une complexité accrue dans la conception de l'algorithme. Il devient nécessaire

de gérer la diversité de la population, pour éviter la dégénérescence qui consisterait à avoir une population constituée de copies d'une solution unique. Il faut aussi assurer une "communication" entre individus, grâce à des méthodes de suivi d'exploration ou de partage de l'espace. Ainsi, des méthodes comme le sharing, ou le clustering adaptatif (voir [WalSmi00] par exemple) permettent d'assurer un bon échantillonnage de l'espace de recherche.

Si tous les problèmes ne sont pas susceptibles de trouver un bénéfice dans un système à solutions multiples, l'approche parisienne permet de résoudre des problèmes mal conditionnés, d'optimiser simultanément plusieurs paramètres, et surtout de faire un usage efficace des évaluations – coûteuses dans la plupart des problèmes d'optimisation – calculées pour *tous* les individus de la population.

Les EA parisiens peuvent intégrer la plupart des caractéristiques des autres EA, mais nécessitent l'introduction de quelques composants supplémentaires pour assurer la qualité de l'*ensemble* de la population :

- deux fonctions de fitness, l'une représentant une évaluation globale de la population, et l'autre mesurant la contribution de chaque individu à la performance globale
- un procédé de répartition de la fitness globale sur chaque individu, en fonction de sa contribution
- un système de maintien de la diversité, permettant de maximiser l'exploration de l'espace de recherche, et l'extraction d'information sur la fonction objectif

Fitness interne, fitness externe

Si l'on recherche non plus un individu mais une population comme solution du problème d'optimisation posé, une simple considération d'espace de définition de la fonction objectif montre que l'usage d'une fitness traditionnelle est impossible : elle est naturellement définie sur le même espace que la fonction objectif, ou un espace des codage pour lequel nous avons défini une fonction de plongement dans l'espace de recherche. Or ici, c'est la population entière qui est un élément de cet espace. Nous ne disposons donc que d'une évaluation globale de la population. Le principe de différenciation des individus, et de reproduction des plus adaptés uniquement, qui est à la base du système évolutionnaire, nécessite quant à lui une évaluation séparée de chaque individu, qu'il nous faut reconstruire grâce à une notion de "participation" à la solution globale, définie heuristiquement.

En réalité, la plupart des problèmes ou est appliquée la méthode parisienne n'entraînent pas une disparition aussi brutale de la fitness individuelle. Il s'agit

en effet de cas où l'on souhaite non plus résoudre un problème d'optimisation simple (à solution unique), mais obtenir l'ensemble des optima, ou l'ensemble des optima locaux. Si l'on se place dans ce cadre, et que l'on considère que les populations sont des approximations de l'ensemble X des optima défini plus haut, il est possible de conserver la fitness individuelle, définie à partir de la fonction objectif.

Cependant, même dans ce cas, il est judicieux de conserver la notion de fitness globale et de fitness locale, cette séparation permettant de procéder à des ajustement de fitness en fonction de la distribution des individus dans l'espace de recherche. Cette méthode permet alors d'intégrer de manière efficace des contraintes mal conditionnées, de garantir une meilleure qualité d'approximation des optima en forçant l'exploration locale. On le verra à la section 4.2, elle est aussi indispensable lorsque l'on souhaite garantir une exploration adéquate de l'espace de recherche.

Elitisme, sélection et remplacement

Les opérateurs de sélection utilisés dans les mécanismes évolutionnaires "classiques" présentés figure 2.5 sont essentiellement dirigés vers la conservation des meilleurs individus comme base d'exploration locale (via les mutations) et comme bibliothèque de matériel génétique performant (via le crossover). Ce procédé s'oppose à l'intérêt porté à l'ensemble de la population dans l'approche parisienne. Elle appelle des opérateurs de sélection plus progressifs, visant simplement à éliminer les individus contribuant le moins à l'élaboration de la solution globale (ou ceux présentant une redondance importante, voir la section suivante).

Une première méthode souvent employée est de réutiliser les schémas de sélection classiques, en augmentant l'élistisme (par exemple, en conservant 20% de la population) et en utilisant des sélections favorisant moins les individus possédant une haute fitness (tournois avec un petit nombre de participants, ou mieux encore ranking). Celle-ci donne souvent des résultats satisfaisants en première approche.

Des schémas de sélection spécifiques, établis en prenant en compte la fitness locale plutôt que la fitness résultante pour établir une liste –restreinte– d'individus devant être remplacés à la prochaine génération, est plus approprié, et tient mieux compte de l'aspect coopératif de l'approche parisienne.

L'utilisation d'un remplacement conditionnel permet de tirer un meilleur parti de ces deux aspects, permettant de conserver les parents si leur contribution est supérieure à celle qu'auraient les descendants obtenus, ou de recommencer le

processus de reproduction tant que c'est le cas.

Partage et diversité

Un élément essentiel au succès d'une approche parisienne... est de garantir que l'on ne parvient pas à une solution de type classique, où tous les individus se concentrent autour de l'optimum global. Plus généralement, on cherche à assurer une exploration optimale de l'espace de recherche, et même dans certains cas à en assurer un échantillonnage optimal à toute les générations (comme dans le cas de l'optimisation multicritère, lorsqu'on cherche à identifier un front de Pareto).

Le maintien de diversité peut s'effectuer grâce à des approches du type nichage ("sharing", comme dans [Mah97] et [Hor97], ou "clustering", comme dans [WalSmi00]). Les approches de type sharing tendent à considérer, en ligne avec l'inspiration darwinienne des AE, que la fitness est une ressource, qui doit être partagée en fonction de la proximité spatiale des individus. Aussi des individus rapprochés auront-ils accès à une "part moins importante" de la fitness que s'ils étaient dispersés.

Le sharing nécessite en tout premier lieu la définition d'une distance entre individus. Les plus simples sont définies comme des distances informationnelles sur le code génétique des individus (Hamming sur des chaînes binaires, par exemple). Il est cependant nécessaire que la distance utilisée corresponde à la topologie effective de l'espace de recherche, très différente de celle fournie par de telles distances dans une grande part des problèmes réels. Le choix de cette distance est en tout cas crucial à la réussite d'une méthode de sharing, et particulièrement difficile dans les cas d'optimisation sur des espaces de recherche non numériques (programmation génétique, par exemple) dont la topologie est très mal connue.

L'implémentation du sharing passe alors par l'introduction d'une pénalité de proximité des individu, en retranchant à leur fitness (locale, dans le cas d'un AE parisien) une quantité d'autant plus grande que la population contient des individus proches. Deux modes de calculs sont principalement utilisés. Le premier consiste à se fixer une fonction t de $\mathbb{R}^+$ dans lui-même, décroissant rapidement, et à calculer une nouvelle fitness $\hat{f}(x) \sum_{i \in \mathcal{P}, i \neq x} t(d(x,i))$. Le second consiste à identifier des sous-populations concentrées, à l'aide d'une méthode de clustering (qui utilise la distance définie pour former des sous-groupes proches d'individus), et à répartir la fitness moyenne du groupe sur l'ensemble de ses individus (par exemple en attribuant comme nouvelle fitness la moyenne divisée par le nombre d'individus dans le groupe).

4.3 Algorithmes évolutionnaires interactifs

Les algorithmes évolutionnaires interactifs se démarquent du cadre défini précédemment pour les AE par le fait que les évaluations des individus sont fournies, au moins en partie, par un opérateur humain. Dès lors il n'est plus possible de considérer que l'on se situe dans le cadre d'un processus d'optimisation tel que nous l'avons défini jusqu'à ce point. De fait, les fitness ne sont pas reproductibles : elles sont bruitées, et variables dans le temps.

A défaut de pouvoir aborder cet aspect sous l'angle théorique (il faudrait pour cela modéliser l'opérateur humain), des résultats pratiques ont montré la validité de la méthode évolutionnaire (et tout particulièrement de l'approche parisienne) dans ce contexte : [Ban01], [Tak98], et récemment [ChaLut01]. Il est par contre nécessaire non seulement de développer des méthodes adaptées pour gérer ces irrégularités, mais aussi d'adapter les procédés de réduction du nombre d'évaluations nécessaires, développées dans le cadre de l'optimisation de fonctions objectif dont le calcul est coûteux (en s'inspirant, par exemple, de [LebLut+98]).

Techniques d'évaluation

Quelle que soit la forme de l'individu à évaluer, une évaluation humaine – valeur numérique associée à une entité présentée par le système, dans le cadre d'une échelle donnée – est soumise à deux sources de variabilité : un bruit, et une dépendance historique.

Le bruit peut avoir des origines diverses : granularité de l'échelle (s'il s'agit par exemple de donner une note de 1 à 10), perception ou étude incomplète de l'objet à évaluer (ayant pour origine l'inattention, ou la confusion d'éléments entre eux), erreur d'évaluation, ou de report de l'évaluation souhaitée. Il est très difficile d'en faire un modèle statistique, car il dépend de l'attention de l'utilisateur, de son environnement, et généralement de facteurs événementiels incontrôlables. Il n'est guère possible que de fixer une amplitude maximale, déterminée empiriquement en fonction du contexte.

La dépendance historique entraîne la modification de l'évaluation d'un même individu en fonction des autres objets présentés par le système entre deux évaluations. Dans le cas d'évaluations "symboliques", portant sur des objets abstraits, cette dépendance est souvent minimisée par la conscience qu'en a l'utilisateur, qui tente souvent lui-même d'assurer une certaine stabilité de son jugement. En revanche, les évaluation sensorielles (couleurs, sons) sont soumises à une dépen-

dance importante. Un exemple académique en est la classification de couleurs présentées successivement en "vert" ou "bleu", un ordre de succession approprié permettant d'induire des classifications multiples de la même couleur, de manière reproductible.

Même si elle reste faible, cette dépendance doit être envisagée avec soin, car elle introduit un biais systématique à même d'orienter l'évolution, contrairement au bruit simple qui ne peut que diminuer la précision des résultats ou retarder la convergence.

Votes, granularité et richesse de l'information

Dans de nombreuses applications des AE interactifs, l'évaluation des individus se fait par l'attribution d'une note sur une échelle entière courte (typiquement, note de 1 à 5 ou de 1 à 10). Cette méthode induit inévitablement une imprécision importante sur les évaluations (20% ou 10% dans les cas précédents). Cependant, cette imprécision a peu d'influence sur la conduite de l'algorithme, car elle reste du même ordre de grandeur que les bruits ayant une source "environnementale". Plus gênants sont les effets de seuil qu'une telle échelle introduit : lors des classement d'individus par fitness, utilisés dans tous les mécanisme de sélection, ces bruits suffisent à permuter des blocs complets d'individus, et donc à les éliminer ou les conserver dans la génération suivante.

Cette limitation, qui apparaît couramment au sein des AEI, peut être contournée en répétant les évaluations, en en utilisant une moyenne. Si cela ne limite que très faiblement le bruit (étant donné le nombre réduit de répétitions qu'il est possible de se permettre dans ce contexte), en revanche cela permet d'obtenir des valeurs non entières, qui diminuent ces effets de seuil. Il est même parfois possible, lorsque la sensibilité de l'algorithme à la qualité des évaluations le permet, de rajouter un bruit gaussien (par exemple) aux évaluations pour éliminer ce seuillage.

4.4 Outils

Au cours des diverses applications et formalisations des AE, depuis les travaux fondateurs de Holland, de nombreux outils spécifiques ont été développés pour les AE. Qu'il s'agisse de codages ayant un usage dans de nombreux problèmes similaires, d'opérateurs de sélection ou de reproduction, ou de méthodes pour gérer certaines contraintes "classiques" pesant sur l'implémentation des algorithmes, ceux-ci ont été éprouvés et améliorés pour s'adapter au plus grand

nombre de problèmes. C'est en outre pour des algorithmes utilisant ces outils que l'on dispose de résultats théoriques permettant de contrôler l'efficacité des algorithmes.

Nous ne tenterons pas ici d'en faire une étude générale, mais de citer ici les outils les plus largement utilisés.

Operateurs génétiques : codages, mutation, crossover

La forme adoptée par les opérateurs génétiques est essentiellement liée aux types de codages sur lesquels ils opèrent. Trois formes servent de base à la plus grande partie des codages utilisés : réel, chaîne de bits, et arbre, ce dernier servant en particulier en programmation génétique.

Codage réel

Le codage sous forme réelle, où le génome est un nombre réel, ou un vecteur de réels (il s'agit bien sûr d'une approximation, la précision étant limitée par les capacités des ordinateurs), est logiquement surtout utilisé lorsque l'ón cherche à réaliser une optimisation sur des espaces numériques. Pour rester bref, l'intérêt par rapport aux méthodes d'óptimisation déterministes, du type suivi de gradient, ou purement stochastiques (Monte-Carlo), se décline en deux aspects. Les AE (on parle souvent dans ce contextes de stratégies d'évolution) sont capables d'optimiser des fonctions objectif très irrégulières, pour lesquelles on dispose de peu d'informations topologiques. Ils permettent aussi de limiter le nombre d'évaluations nécessaires (les méthodes de suivi de gradient sont très coûteuses sur ce point), lorsque le calcul de la fonction objectif est long.

Les opérateurs associés sont essentiellement d'inspiration numérique. La mutation est l'ajout d'un bruit (souvent gaussien, il existe des résultats s'appliquant à ce type de mutation, en application de [Cer95] et – partiellement – de [YLSLut00]). Le crossover peut être un simple barycentre, à coefficients fixés ou aléatoires, ou bien un crossover bi-parental, dont le résultat est déterminé comme l'optimum dans un échantillonnage de la droite joignant les parents.

Codage binaire

Les chaînes de bits (historiquement, parmi les premiers types de génomes utilisés) ont été employés pour représenter des notions très diverses (y compris des nombres réels...). En conséquences les opérateurs développés pour ces codages

sont très variés. Beaucoup s'inspirent cependant des méthodes de recombinaison de génome à l'oeuvre lors de la reproduction naturelle. La mutation se traduit par une inversion de bits ("flip", passage de 0 à 1 et de 1 à 0) déterminés aléatoirement. La distribution est souvent uniforme sur la chaîne de bits, et on se contente donc de la caractériser par la probabilité d'inversion de chaque bit de la chaîne, typiquement faible, de l'ordre de 1%. Le crossover est une recombinaison de morceaux de chaînes de bits. Cela se traduit par le choix aléatoire de points de coupure sur le génome des deux parents (éventuellement distincts lorsque le génome est de longueur variable) et l'échange de sections entre ces points de coupure. Pratiquement, il est notoire que les performances des AE peuvent être considérablement améliorées par l'introduction de connaissances liées au problème (topologie de l'espace de recherche, par exemple) dans les opérateurs génétiques, et tout particulièrement via le crossover. Aussi ces formes "typiques" ont donné lieu à de nombreuses variations spécifiques des problèmes auxquels elles ont été appliquées.

Codages en arbres

Les codages sous formes d'arbres, quant à eux, correspondent aux approches du type programmation génétique, dont nous parlerons en détail plus bas. Ils se démarquent des deux précédents par la complexité des opérateurs mis en oeuvre, qui traitent le génome plus globalement que dans les cas réel et binaire. Les principaux aspects qui déterminent la forme de ces opérateurs est l'arité – souvent fixée – des noeuds, et la nécessité de limiter une croissance immodérée des branches des arbres (le phénomène de "bloat"). Les mutations les plus courantes sont le remplacement d'un élément terminal par un autre, l'interversion de deux sous-arbres (éventuellement, enfants de noeuds différents), et l'insertion ou la suppression de sous-arbres, lorsque les noeuds ne sont pas d'arité fixe. Le crossover est généralement un échange de feuilles ou de sous-arbres.

Lorsque la structure des arbres est contrainte (arbres équilibrés, par exemple), il est courant d'intégrer dans ces opérateurs une fonctionnalité de "réparation", permettant de transformer une structure invalide en un arbre valide (par suppression de sous arbres, insertion de sous-arbres générés aléatoirement, ou déplacement de ceux-ci).

Enfin, il convient d'ajouter à cette liste des techniques d'élagage, permettant de limiter le bloat par suppression de sous-arbres trop longs (pouvant s'écrire sous une forme plus concise), ou bien qui ne sont pas utilisés (enfants surnuméraires d'un noeud dont l'arité réelle est plus faible).

Faisabilité, pénalités

Nous avons évoqué brièvement plus haut la question de l'existence de codages qu'il est impossible de traduire dans l'espace de recherche. On parle alors d'individus infaisables. Cela peut provenir de la définition de l'espace de recherche comme une réduction d'un espace plus vaste par des contraintes qui ne peuvent être formulées au niveau des codages, ou bien de l'impossibilité d'évaluer la fonction objectif sur une partie de l'espace de recherche.

Il est parfois possible de supprimer ces individus, en construisant les opérateurs génétiques de manière à ce qu'ils ne produisent que des individus faisables. Dans le cas d'un codage numérique par exemple, cet effet peut être obtenu par projection sur les sous-espaces "admissibles". Dans le cas de l'optimisation sous contraintes, lorsque les contraintes sont mal conditionnées ou séparent l'espace de recherches en parties non connexes (du point de vue des opérateurs génétiques), il est en revanche souvent souhaitable de conserver ces individus infaisables dans la population, de manière à obtenir une meilleure exploration de l'espace dans les zones difficilement accessibles (parties non connexes, ou discrètes, par exemple). Il est cependant nécessaire de limiter leur apparition, pour orienter l'évolution vers des codages admissibles. La méthode adoptée est le plus souvent une pénalité sur la fitness infligées aux individus infaisables, prenant éventuellement en compte la distance aux régions admissibles lorsqu'il est possible de la définir.

4.5 Programmation Génétique

La programmation génétique est un cas particulier d'algorithmes évolutionnaires, qui se propose de réaliser une optimisation dans un espace de programmes. Il existe peu de différence intrinsèques entre les deux méthodes (il s'agit d'interprétations entièrement compatibles du paradigme darwinnien). La principale différence est historique : les premiers AG traitaient des individus codés par des chaines binaires de taille fixe, tandis que la PG traite des individus de taille variable.

Faire évoluer des individus de taille variable et très structurés impose cependant d'utiliser des techniques bien différentes de celle employées dans les AG classiques. Parmis les problèmes qui se posent arrivent en premier les questions de croissance de la taille des individus ("bloat", voir [LanPol97]), la question de la diversité, déjà abordée plus haut, la sauvegarde de caractères évolués (notions d'introns, selon [WalSmi00]). Surtout, disposer d'un codage efficace des individus et d'opérateurs génétiques précisément adaptés aux type de problème à

traiter devient primordial.

De nombreux types de codages ont été utilisés en PG. Les plus courants restent cependant les codages à bases d'arbres, que nous avons déjà évoqués. Ils disposent de l'avantage considérable de fournir un mode de représentation adapté à une très grande classe de problèmes : l'optimisation dans des espaces de fonctions vectorielles, comme la régression symbolique, les problèmes inverses numériques, et de nombreux cas de modélisation contrainte. En outre, les opérateurs évolutionnaires sur des arbres sont bien étudiés, et l'on dispose de résultats théoriques sur de telles structures.

Cependant, les arbres ne suffisent pas toujours, ou ne sont pas toujours la représentation la plus pratique. Lorsque l'on fait appel à des structures plus complexes, le principal problème est la conception d'opérateurs génétiques traitant de telles structures efficacement, et conservant les attributs produits par l'évolution. Des techniques telles que lévolution grammaticale (voir [NeiRya98a, NeiRya+98b]) simplifient ce problème, en le décomposant. D'une part, la structure des attributs que l'on souhaite voir apparaitre est décrite au moyen de règles de production (description sous forme Backus Naur). D'autre part, l'évolution est réalisée sur des chaînes binaires de longueur variable. Chaque mot (groupe de bits de taille fixée) de ces chaînes déclenche l'activation d'une règle de production, permettant la transcription des chaînes en programmes. Toute forme de code (y compris C, C++, assembleur...) peut être utilisée dans les règles de production, ce qui permet de s'affranchir des contraintes dues au codage.

Chapitre 3

ELISE : UN OPTIMISEUR DE PROFIL DE RECHERCHE

Résumé du Chapitre.

Nous avons vu Chapitre 2 que la méthode évolutionnaire présente de nombreux avantages pour réaliser l'optimisation de profils utilisateurs, problème intrinsèquement difficile.

ELISE, pour "*Evolutionary Learning Search Engine*", est un prototype de moteur de recherche personnalisé permettant d'intégrer les techniques décrites plus haut à l'outil évolutionnaire, pour réaliser des tests tant sur des bases documentaires de référence qu'en vraie grandeur.

Diverses stratégies de paramétrage des AE parisiens nous permettent de renforcer leur capacité à utiliser efficacement des évaluations bruitées : faibles taux d'application des opérateurs génétiques, conservation d'une grande part des populations à chaque génération, et accumulation des mesures sur plusieurs générations. Réaliser une génération pour chaque requête utilisateur permet en outre de conserver une bonne réactivité à un changement de comportement de l'utilisateur.

ELISE propose une interface très à celle des outils de TR classiques : langage de requêtes booléen, et présentation des résultats sous forme de liste. Ceci est nécessaire pour assurer une prise en main rapide et efficace de l'outil, et par là favoriser la bonne qualité des mesures.

De fait, Elise peut être envisagée comme une "couche" supplémentaire adap-

table sur des outils de TR déjà installés. Elise réalise une personnalisation du traitement des requêtes par ces outils, grâce à un profil utilisateur. Celui-ci est optimisé en relation avec une mesure de satisfaction, tirée du temps passé par l'utilisateur sur les documents proposés. La spécification d'Elise à un niveau assez général, à partir de filtres et de flux de donées, permet une construction modulaire et une bonne intégration dans le contexte des réseaux de données.

L'optimisation des profils repose sur une approche de programmation génétique. De tels espaces de recherche non numériques rendent difficile la définition d'une topologie adaptée au problème, aussi des informations complémentaires – ici sémantiques – permettant d'orienter la recherche sont nécessaires. Il est aussi important de tenir compte de problèmes classiques de cette approche : bloat, obsolescence des individus, historique des évaluations.

Pour ne pas être amené à faire des hypothèses cognitives risquées sur le comportement des utilisateurs, nous ne tentons pas de produire de modèle. Les profils codent simplement la réponse du moteur aux actes de l'utilisateur. Leur format doit se plier aisément aux outils éprouvés de la PG, et pour autant réaliser efficacement le traitement de requêtes volumineuses et complexes. Ces deux contraintes nous ont amené à développer un langage dédié au codage de ces profils : OKit.

OKit est un langage concaténatif à pile, assez générique pour être utilisable dans la plupart des contextes de PG. La spécificité au TR provient de la définition des instructions, briques élémentaires du langage. Une grande partie des opérateurs classiques sur des arbres utilisés en PG sont transposables. Une attention particulière doit cependant être accordée à l'obtention d'individus dans l'espace des solutions admissibles, autrement dit dont l'évaluation ne produit pas d'erreur.

1 Méthodologie : AG et approche parisienne interactive

Rappelons le cadre que nous nous sommes fixé : il s'agit d'optimiser un profil contenant les informations nécessaires à un traitement personnalisé des requêtes, en fonction d'une mesure de satisfaction de l'utilisateur portant sur les résultats de recherche obtenus grâce à ce profil.

Réaliser un profil permettant d'adapter l'outil de recherche aux comportements de l'utilisateur, sans établir un modèle explicite de ceux-ci, est un problème d'optimisation très difficile.

Le profil permet de traiter de manière personnalisée les requêtes de l'utilisateur. Le résultat de ce traitement permet alors d'obtenir une liste de résultats extraits de la base documentaire, qui seront évalués par l'utilisateur. Ainsi, nous obtenons pour chaque requête une liste de documents dont l'utilisateur indique qu'ils constituent des réponses acceptables à sa requête. Nous sommes alors confrontés à un problème inverse implicite, consistant à trouver la méthode de traitement de cette requête produisant la liste de documents acceptables, ou s'en approchant au plus près.

De plus, les optima recherchés sont multiples (de nombreux modes de traitement peuvent amener à des résultats également satisfaisants), et l'objectif de l'optimisation change au cours même de l'optimisation, selon les intérêts et opinions de l'utilisateur.

L'espace des profils peut être décrit très précisément de manière algorithmique, ce qui fournit une topologie intrinsèque très structurante. Cependant, celle-ci n'est pas adaptée au problème. Nous ne disposons pas d'une connaissance suffisante de la base documentaire, ni d'une notion assez précise de ce qui constitue deux documents similaires pour l'utilisateur, pour être en mesure de construire une topologie adaptée sur ces espaces de programmes.

La méthode évolutionnaire interactive semble donc la plus adaptée pour la construction des profils utilisateurs, a cause de sa flexibilité, sa robustesse, et sa capacité à travailler dans des espaces "amorphes", sans topologie significative. Mais en tirer parti efficacement impose un examen complet des conditions dans lesquelles elle sera mise en oeuvre, et des moyens à notre disposition.

1.1 Interaction et évaluation

La contrainte primaire portant sur la mise en oeuvre d'un AG dans ce contexte est donc l'évaluation de cette satisfaction, qui fournit l'élément moteur de l'optimisation : la fonction objectif, encore appelée fitness dans le cadre des AG. Celle-ci est ici acquise auprès de l'utilisateur du système.

Dans la plupart des applications des AG, où la fitness est issue d'un calcul numérique, elle constitue le principal coût algorithmique. On doit alors éventuellement se préoccuper du nombre d'évaluations nécessaire, pour tenter de le limiter au moyen d'approximations, en réutilisant de précédentes évaluations, ou encore en limitant le nombre d'individus à évaluer (voir par exemple [LebLut+98]). Les facteurs limitants de l'évolution sont alors la précision et la disponibilité des évaluations de fitness, qui doivent être judicieusement équilibrés avec le coût des calculs pour obtenir un algorithme performant [LutCol02].

Pour des raisons très différentes, les facteurs limitants sont ici les mêmes. Chaque évaluation est le résultat de l'interrogation de l'utilisateur, qu'elle soit explicite (demande d'une "note") ou implicite (observation du comportement). Dans les deux cas, un nombre trop important de demandes est source de lassitude, et nombre de facteurs externes, que nous ne contrôlons pas, influent sur l'évaluation. Les variations d'attention de l'utilisateur, en particulier, influent sur la qualité de celles-ci.

Par contre, il n'y a pas ici d'équilibre à trouver : nous devons simplement nous contenter du nombre et de la qualité des évaluations disponibles, qui dépendent de la personnalité et de la patience de chaque utilisateur. Il faut donc tirer parti au mieux de ces informations.

Fitness bruitée, variable

Le premier facteur limitant, la précision, ou plus généralement la qualité, de la fitness, est comme on l'a vu plus haut inhérent à tout algorithme interactif. L'évaluation de résultats de recherche est particulièrement sujette à incertitudes.

D'une part, l'appréciation par l'utilisateur de la pertinence d'un résultat – ou d'un document en particulier – dépend de l'historique de recherche. Les informations acquises au cours des précédentes recherches peuvent modifier la perception qu'il a de la relation de certains sujets aux résultats obtenus, ou même du sens de certains termes. La reproductibilité de l'évaluation n'est donc pas assurée, et le problème à résoudre n'est pas réellement une optimisation statique, mais un suivi d'optima. De plus, nous ne pouvons faire l'hypothèse de la continuité, de

toute manière peu significative en l'absence d'une structure connue de l'espace documentaire. Il est donc impossible d'assurer qu'un précédent profil performant le sera encore pour la prochaine requête, et nous devons être prêts à gérer des transitions brutales dans le paysage de fitness.

D'autre part, une évaluation de résultats de recherche repose sur une interprétation de textes, comportant inévitablement une part importante de subjectivité. C'est celle-ci qui est à l'origine des limitations actuelles rencontrées par les méthodes heuristiques ou statistiques pour la recherche textuelle [1]. La fitness est elle-même soumise à cette subjectivité, qui engendre des variations provenant d'un bruit systématique, et non d'une modification du paysage de fitness. Si une analyse statistique permettrait de former un modèle de ce bruit, les caractéristiques numériques obtenues dépendraient tant de l'utilisateur que du sujet étudiés, et seraient donc de peu d'utilité pratique. Nous pouvons tout de même admettre l'hypothèse – peu contraignante – de l'applicabilité d'un théorème central limite. Si cela ne permet pas de fonder un traitement statistique des comportements de recherche, cette hypothèse valide en revanche la limitation du bruit sur les évaluations par leur accumulation sur plusieurs requêtes.

Cependant, cette approche nous oblige à augmenter le nombre d'évaluations demandées à l'utilisateur, ce qui, on l'a dit plus haut, n'est pas souhaitable. Plus grave, elle entrave la mise en œuvre des solutions éventuelles à la première source d'imprécision. Mettre sur le même plan des évaluations obtenues pour des instants différents, alors même que la fonction objectif globale (évidemment inconnue) a changé, fait obstacle au suivi efficace de ces changements. Ici encore, un équilibre est à trouver, en adaptant le nombre d'évaluations accumulées aux caractéristiques des utilisateurs visés et de la base documentaire explorée.

Modèle AGI synchrone

Le second facteur limitant est la disponibilité des évaluations. Le "grain" le plus fin qu'il est possible d'obtenir dans l'évaluation des résultats de requêtes est une estimation de la pertinence de chaque document présenté dans les listes de résultats. Des études statistiques[2] montrent que seuls les 20 premiers résultats sont examinés par les utilisateurs les plus attentifs (et seulement les 3 premiers pour certains), ce qui limite les évaluations à une dizaine par requête.

De plus, ces évaluations ne sont disponibles que dans le contexte d'une requête, ce qui limite leur pertinence au profil utilisé pour celle-ci. Leur utilisation

[1] ... et a donc indirectement amené cette recherche !

[2] études internes d'usage des outils de TR menées au sein de Novartis Pharma.

pour diriger l'optimisation doit donc être immédiate.

Ces éléments nous ont amené à envisager un modèle d'algorithme génétique interactif synchrone. Lors de chaque requête, une nouvelle génération est produite, fournissant une nouvelle version du profil utilisateur. Celui-ci est utilisé pour traiter la requête, et la liste de résultats est présentée à l'utilisateur. Nous obtenons alors une évaluation des individus composant le nouveau profil, qui sera utilisée pour produire la prochaine génération.

1.2 Interface avec les outils de TR classiques et l'utilisateur

Les outils de text-retrieval classiques ont été développés pour le traitement efficace de données de types très différents (les formats de documents sont nombreux) et la recherche rapide dans les bases documentaires. Il s'agit d'outils complexes, apportant des services très importants (en termes de recherche aussi bien qu'en termes d'accès aux documents). Ils sont souvent associés de manière étroite aux bases documentaires, au point de former des systèmes de gestion de l'information à part entière. Réutiliser ces outils est la condition du succès de tout outil de TR. Les méthodes développées ici doivent donc être envisagées comme une extension de ces outils, et non un remplacement.

En ce qui concerne l'interface, beaucoup d'utilisateurs ont déjà accès à un outil de TR, ou au minimum aux moteurs de recherche sur internet, et ont déjà une très bonne habitude de l'outil. Tous ces logiciels partagent des modes d'interaction et une présentation similaires, établissant de fait un standard. A ce stade, changer l'apparence et la réactivité du système impliquerait une phase d'adaptation, voire un rejet.

Le système d'optimisation par AG doit donc s'accommoder de deux interfaces :

- celle avec l'utilisateur, dont il tire ses objectifs,
- celle avec les outils de TR existants, dont nous ferons l'hypothèse ici qu'il s'agit d'un moteur de recherche booléen[3].

Dans les deux cas, il s'agit de s'accommoder des systèmes existants, pour lesquels le nombre important d'outils disponibles et leur efficacité nous sont précieux.

[3]Ce mode de recherche est disponible dans la plupart des solutions de TR commerciales, de fait elles sont souvent capables de modes de recherche beaucoup plus élaborés, que nous n'avons pas cherché à utiliser ici tant pour des raisons d'universalité que parce que ces solutions élaborées ne nous offrent pas pour l'instant un contrôle suffisant sur le processus de recherche.

Requêtes, listes de résultats

Comme nous venons de le voir, la connaissance préalable des possibilités de l'interface, et l'adaptation en un temps réduit de l'utilisateur sont des éléments essentiels de la bonne acceptation du système et de son utilisation efficace. Aussi avons-nous conservé une apparence et un mode de formalisation des requêtes très similaire à ceux rendus familiers par les moteurs de recherche internet.

L'interface présente tout d'abord une page permettant de taper une requête, sous la forme d'une liste de mots, ou optionnellement une requête booléenne (voir Chapitre 1, section 1.2), désignant les notions que l'utilisateur souhaite voir apparaître dans les documents renvoyés. En réponse parvient une page de résultats listant les résultats sous la forme d'un titre cliquable agrémenté de quelques lignes informatives. La réponse à une requête apparaît ainsi sous une forme proche de celle présentée figure 3.1.

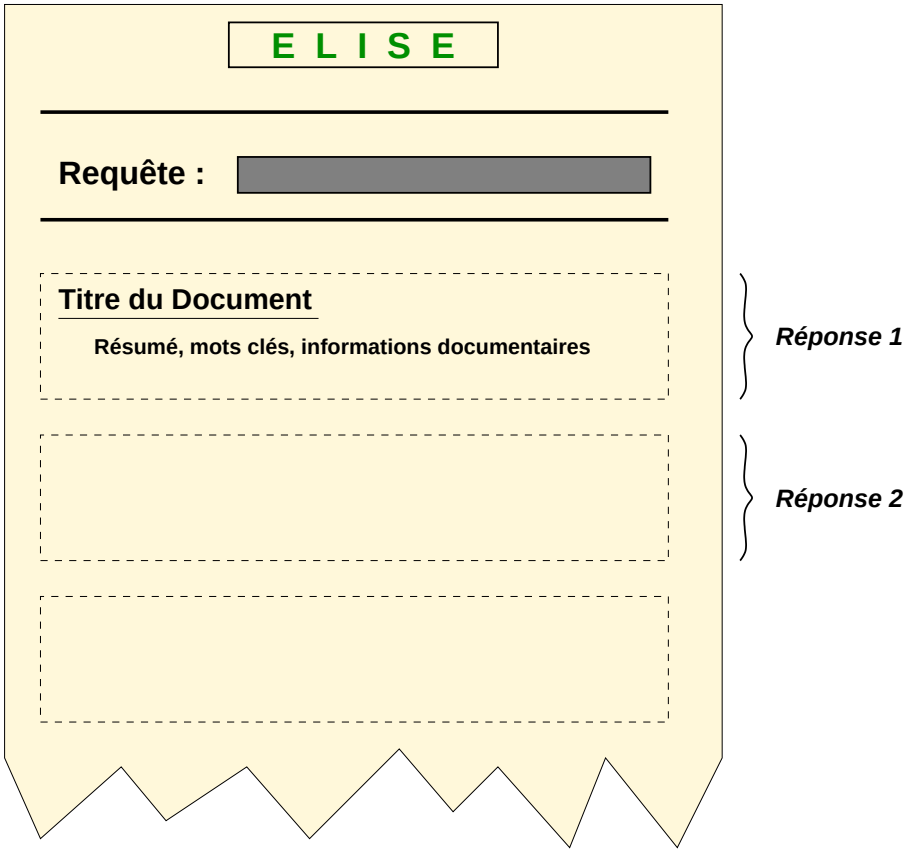


FIG. 3.1 – Interface : vue globale.

Cliquer sur les titres des documents permet d'accéder aux documents originaux eux-mêmes, accès qui seront enregistrés, fournissant la base des évaluations utilisées lors de l'apprentissage. Les informations accompagnant chaque document permettent de rendre ces évaluations plus fiables, favorisant une sélection réfléchie, "informée", des documents visualisés. Elles peuvent être issues d'une extraction sémantique dirigée par les termes de la requête, des champs "résumé" s'il en existe dans les documents eux-mêmes, ou de toute autre type d'information fourni par des outils préexistants. Il est cependant souhaitable de consacrer à ces informations un espace limité, ce qui permet, en un coup d'œuil, de trier un nombre de réponses plus important. Cela permet, à la fois, de diminuer le temps nécessaire à l'obtention des résultats pertinents, et favorise la consultation d'un nombre important de documents, ce qui augmente le volume de données disponibles pour l'apprentissage.

Un élément déterminant pour la performance du système et la satisfaction des utilisateur est l'ordre de présentation des résultats. Les plus pertinents doivent impérativement être présentés en tête de liste. Or, comme nous l'avons amplement analysé plus haut, établir la pertinence d'un document vis-à-vis de la requête formulée est très difficile, voire sujet à de nombreux contresens. Les outils de TR utilisés dans les moteurs de recherches web utilisent des mesures "simples" de pertinence, comme une "corrélation" empirique entre termes de la requête et termes présents dans le document. Si celles-ci sont efficaces pour un grand nombre de recherches peu spécifiques, leur échec est spectaculaire lorsque des termes rares ou des notions mal cernées par les bases de vocabulaires interviennent (conduisant à un ordre de présentation des documents au mieux aléatoire, au pire contraire aux attentes).

La sofistication des méthodes de classement (dites de "ranking") des documents dans les ensembles de résultats est un argument de prescription crucial dans la plupart des solutions commerciales. Les sytèmes de classement plus complexes, faisant parfois appel à des systèmes d'apprentissage, ne manquent donc pas. Nous avons cependant préféré nous concentrer sur le contenu des listes de réponses, problème beaucoup moins exploré. Le soin du classement des listes de résultats sera délégué aux outils préexistants des solutions de TR utilisés, dont la performance est très souvent adéquate.

Extraction des documents

Nous l'avons vu, les outils de TR modernes, pour pouvoir gérer les tailles de bases documentaires courantes dans des environnements industriels, et les volumes de requêtes correspondants, utilisent des technologies d'accès et de

traitement des données élaborées et complexes. Si les nombreuses années de développement investies dans la plupart des solutions commerciales (expliquant le coût élevé des licences) ont été consacrées en partie à la mise au point d'heuristiques d'expansion des requêtes et de méthodes d'indexation plus fines des documents, il reste que construire un système performant d'indexation et d'accès aux documents demande temps et expertise.

Réutiliser les outils de TR préexistants nous permet de tirer parti de ces systèmes optimisés, des structures correspondantes d'archivage et d'indexation mises en places pour gérer les bases documentaires sur lesquelles doit s'appuyer tout test en vraie grandeur de cette recherche. En revanche, tant pour assurer la reproductibilité des résultats dans une phase de test que pour éviter tout biais contraire aux attentes des utilisateurs, il est nécessaire de s'affranchir des heuristiques et outils d'analyse lexicale mis en place dans ces systèmes. Heureusement, la plupart d'entre eux utilisent des index par termes, et disposent d'un mode de recherche booléen, ne réalisant aucune expansion ou modification de la requête. Il est ainsi possible d'accéder au coeur des fonctionnalités d'accès à la base documentaire et d'examen des index.

Ce mode de recherche booléen demande en entrée une expression formée à partir d'opérateurs logiques (voir Chapitre 1, section 1.2), et fournit en retour une liste de documents validant cette expression. L'utilisation de ce mode d'interrogation impose dès lors une contrainte fonctionnelle sur les profils utilisateurs : ceux-ci doivent permettre de transformer une requête de l'utilisateur (liste de mots, ou expression booléenne) en requête booléenne stricte (les opérateurs omis étant remplacés par un opérateur "implicite" dépendant du moteur de recherche, le plus souvent **#AND**).

La liste de documents fournie en réponse comporte (dans tous les outils que nous avons examinés) deux informations complémentaires dont nous ferons usage : un "rang" du document vis à vis de la requête (évaluation de la pertinence du document, basée sur des heuristiques variant largement d'un outil de TR à l'autre), et un "extrait", ou résumé. Le premier nous permettra de trier les documents dans les listes de résultats que nous proposerons à l'examen de l'utilisateur, et le second tiendra lieu d'information sur le document.

Réponses, votes

Pour "apprendre" les profils utilisateur, ou, pour changer de paradigme, pour trouver le profil optimal, il nous faut une évaluation de la qualité de chacun des profils, la "fonction objectif" du problème d'optimisation qui nous est posé.

Cependant, ici, il s'agit d'optimiser la "satisfaction" de l'utilisateur, quantité d'autant plus difficile à calculer que sa définition même est délicate. Demander directement à l'utilisateur d'indiquer son niveau de satisfaction, *à chaque recherche*, requérant un niveau de participation incompatible avec l'esprit des moteurs de recherche (un outil simple devant se faire oublier), nous devons l'évaluer de manière détournée, par l'observation et l'analyse des comportements.

Comme toute analyse comportementale, celle-ci repose sur des hypothèses d'ordre psychologique. Nous supposerons donc que la majorité des utilisateurs est capable d'identifier rapidement la pertinence d'un document à sa lecture, que le temps passé à lire le document est fonction de son intérêt, et que, dans une liste de documents, quelques uns des premiers documents intéressants seront visités. Si ces hypothèses reposent sur l'observation de quelques utilisateurs, nous ne disposons pas de données statistiques fiables permettant de les valider. Des études de cet ordre on très probablement été effectuées par les principaux éditeurs de solutions de TR, mais elles n'ont à notre connaissance pas été publiées, et nous n'avons pas été en mesure d'en réaliser de semblables.

Dans le cadre d'une optimisation par un AG "synchrone", pour lequel une génération est produite à chaque requête, l'élément atomique sur lequel peut porter notre évaluation est chaque document renvoyé. Si chacun d'eux (chaque entrée dans la liste de résultats) est accompagné d'un texte informatif, celui-ci n'est pas suffisant pour assurer que les documents visualisés sont forcément pertinents pour l'utilisateur. En revanche, si l'utilisateur est capable de faire une analyse rapide de cette pertinence à la lecture du document, décompter le temps passé à cette lecture fournit une évaluation fiable. Nous disposons ainsi d'une évaluation atomique des résultats, construite en deux temps : un document non visualisé **n'est pas** pertinent, et le degré de pertinence d'un document visualisé est fonction du temps passé par l'utilisateur sur ce document[4].

Ces données atomiques doivent encore être composées pour fournir une évaluation des individus composant le profil, en fonction des caractéristiques internes de l'AE mis en place.

Une organisation basée sur des flux de données

A ce stade, nous avons décrit (plus ou moins complètement, et de manière explicite ou via une liste de contraintes) une grande part des composants nécessaires à la conception d'un système d'apprentissage des profils. Récapitulons :

[4]Suivre les documents imprimés serait une bonne indication complémentaire de l'intérêt du document, mais n'est pas possible dans le cadre d'une application sur intranet.

- **Profil personnalisé**, transformant une requête de l'utilisateur en requête booléenne.
- **Système de recherche booléen**, répondant à une requête booléenne par une liste de documents validant cette requête.
- **Interface utilisateur**, permettant le suivi des documents visualisés au sein d'une liste de documents.
- **Algorithme évolutionnaire**, utilisant une évaluation de la pertinence des résultats de recherches pour mettre à jour le profil.

Cette liste fait apparaître un circuit en boucle des informations, classique des systèmes d'apprentissage. Si cela n'a rien de surprenant, cet élément facilite grandement l'intégration des technologies développées pour le TR et l'extraction sémantique. En effet, conçues pour s'intégrer dans les environnements client-serveur généralement associés aux intranets et aux bases de données, nombre d'entre elles fonctionnent comme des filtres, transformant des informations fournies en entrée en un flux de sortie. La réalisation du système peut ainsi être vue comme l'enchaînement de plusieurs de ces filtres, de la requête de l'utilisateur aux évaluations récoltées, permettant de transformer chaque requête en valeurs de fitness pour un profil.

1.3 Approche parisienne et AGI synchrone : contenu des profils

Les évaluations de pertinence récoltées à chaque recherche, n'ont valeur que dans le cadre global de la liste de documents renvoyée. Faire le lien entre l'évaluation d'un document et l'évaluation d'un individu de l'algorithme évolutionnaire est difficile, quel que soit le sens que nous donnons à un individu particulier, car il est nécessaire de replacer l'évaluation de pertinence dans le contexte de la liste de résultats. Un "mauvais" document peut ne l'être que parce que l'ensemble de la liste contenait des résultats pertinents, dont un nombre limité a été visualisé, et inversement.

Nous cherchons à optimiser un profil utilisateur, permettant la transformation des requêtes de l'utilisateur en requêtes booléennes. Une approche évolutionnaire classique nous amènerait à concevoir une population de tels profils, pour lesquels nous devrions obtenir des évaluations individuelles. Un très petit nombre de ceux-ci étant performants à chaque génération, nous serions conduits à demander à l'utilisateur d'évaluer des profils de faible qualité, voir générés aléatoirement à la précédente génération, ce qui n'est évidemment pas acceptable.

Ces éléments plaident en faveur de l'approche évolutionnaire parisienne, dans laquelle le profil utilisateur est codé sur une population entière. Dans ce cadre,

l'évaluation individuelle (fitness locale) se place naturellement en rapport avec une évaluation globale, et il n'est pas nécessaire de disposer d'évaluations individuelles indépendantes. De plus, nous disposons alors à chaque génération d'un profil (constitué de la population entière) performant.

L'évolution très progressive liée au modèle parisien permet aussi de conserver sur plusieurs générations des individus (que l'on peut voir comme une "distillation" des informations récoltées sous la forme d'un modèle partiel de l'utilisateur). Il est ainsi possible d'accumuler des évaluations portant sur des requêtes très différentes, favorisant la généralité des profils évolués et diminuant l'influence du bruit, environnemental ou comportemental.

Acquisition d'informations

Dans l'esprit de l'intelligence artificielle symbolique, dont les contributions aux algorithmes évolutionnaires sont nombreuses, de nombreuses méthodes ont été proposées pour tirer parti d'informations non traduisibles sous forme d'évaluation numérique, afin d'orienter l'évolution. Beaucoup sont liées au réglage dynamique des paramètres de l'algorithme génétique (probabilité et distribution des opérateurs, méthodes de sélection, etc... – voir [Dav89, SchMor87] par exemple). Certaines de ces approches ont conduit aux techniques dites "méta", chargeant un autre algorithme d'optimisation (un AE le plus souvent) de l'optimisation des paramètres de l'algorithme évolutionnaire. Si ces dernières ne sont pas applicables dans notre cas (la quantité d'évaluations dont nous disposons est très limitée), des méthodes visant à adapter la sémantique même des opérateurs, dans l'esprit de [SebRav+96] et [RatSeb00], semblent applicables.

En effet, les outils d'analyse sémantique permettent d'obtenir, avec des temps de calcul raisonnables, les liens sémantiques qu'entretiennent les termes des requêtes et leur contexte sémantique dans les documents visualisés. Il est possible d'utiliser simplement ces informations, en construisant et en utilisant un thesaurus spécifique de chaque utilisateur, constitué à partir des termes apparaissant dans les requêtes, et mettant en relation de synonymie les termes proches dans des documents visualisés. Ce thesaurus peut compléter l'usage de thesauri spécifiques au sein des opérateurs génétiques. Ce point sera développé plus en détail section 3.3.

Historique, latence et pérennité des individus

L'approche parisienne permet de conserver certains individus pendant de nombreuses générations, accumulant ainsi les évaluations et permettant de vérifier leur généralité. Pour profiter de cet aspect, il convient cependant de donner un sens effectif à la notion d'"accumulation". Si pour simplifier on acquiert à chaque génération une nouvelle évaluation d'un individu, tant que celui-ci est conservé, cette notion se traduit par l'utilisation, dans le calcul de la fitness, des évaluations passées, en plus de celle récoltée à la génération courante. Evidemment, l'évaluation courante doit avoir le rôle le plus important, afin de réagir efficacement à un changement d'attitude de l'utilisateur (modification de ses centres d'intérêts, par exemple). Une évaluation très mauvaise mais isolée ne doit pas non plus désavantager un individu, en comparaison d'individus possédant un nombre réduit d'évaluations très bonnes : une simple pondération des évaluations passées avec une décroissance temporelle des poids ne suffit pas.

Il faut aussi prendre garde aux effet négatifs d'un tel historique sur l'adaptabilité et la diversité des populations. Conserver un individu sur plusieurs générations implique l'utiliser plusieurs fois comme parent de nouveaux individus. Les opérateurs génétiques des AE parisiens sont construit pour conserver une grande partie des caractéristiques des parents dans le génome des descendants, et la fréquence des mutations est maintenue à un niveau réduit. Une mémoire importante représente donc une menace pour la diversité des populations, et augmente le nombre de génération nécessaires pour réagir à des modifications dans le paysage de fitness.

2 PG, algorithmie et codage

Le codage choisi pour les génomes est un élément important dans la conception d'un système à base d'algorithmes évolutionnaires, et détermine directement sa performance, tant du point de vue de l'efficacité de l'optimisation que du coût algorithmique. Il s'agit ici de coder des profils (ou plutôt, des composants d'un profil), à même de réaliser la traduction d'une requête (booléenne ou exprimée sous forme de liste de termes) en une requête booléenne.

Il est bien sûr possible d'imaginer des méthodes de transformation codées par une simple liste de paramètres numériques, en indiquant l'ordre et/ou la probabilité d'application de procédures de réécriture élémentaires (échange de termes, expansion d'un terme particulier en utilisant des thesauri, etc...). Cet arsenal de procédures forme cependant un modèle implicite des utilisateurs (ce

que nous avons choisi d'éviter), et ne permet en tout cas pas des transformations complexes basées sur une analyse de la structure de la requête.

Le seul outil qui semble pouvoir traiter une requête dans toute sa généralité, sans hypothèse restrictive sur sa structure ou le type de transformations possibles, repose sur l'évolution de programmes de réécriture, autrement dit une approche de type programmation génétique. Il nous faut donc être en mesure de coder un programme, possédant des propriétés adéquates :

- être capable d'opérer sur toute requête pouvant être produite par un utilisateur,
- permettre de définir des opérateurs génétiques efficaces,
- conserver un coût d'exécution raisonnable.

2.1 Contraintes sur le langage

Codages efficaces

Un très grand nombre d'applications de la programmation génétique repose sur des codages à base d'arbres. Ceux-ci ont en effet beaucoup d'avantages : la programmation des structures correspondantes est assez simple, leur exécution est rapide et ils permettent de représenter une large variété de modèles. En particulier, toute formule mathématique de type fonctionnel est représentable de cette manière, pourvu que l'on définisse de manière adéquate un ensemble d'opérateurs et de fonctions élémentaires (noeuds de l'arbre). De telles caractéristiques font de ce type de représentation l'outil de choix dans une application importante de la PG : la recherche de modèles mathématiques d'un ensemble de données (en particulier la régression symbolique, voir par exemple [Koz92]).

A cela s'ajoute la possibilité de coder de manière simple et intuitive les opérateurs génétiques standards. Les mutations se traduisent par des déplacement, créations ou suppressions de sous-arbres, tandis que le crossover utilise des échanges de sous-arbres entre parents. La "boite à outils" génétique est simple, transposable aux différents cas particuliers dont on peut avoir besoin (arbres d'arité fixée ou variable, arbres équilibrés, existence ou non de branches ignorées, ...) et même extensible à des structures similaires, comme les graphes acycliques dirigés (DAG).

Le grand nombre d'applications de ce codage est à l'origine d'un nombre également important de résultats expérimentaux apportant des solutions aux problèmes classiques de la PG : diversité, bloat, paramétrage (tailles de population, de génome, méthodes de sélection et taux d'application des opérateurs

génétiques – voir section 2.3 et chapitre 4). En outre, de nombreux outils (opérateurs spécifiques, méthodes d'évaluation, de génération et de réécriture des arbres – voir section 3.2) ont été mis au point et testés.

Par contraste, d'autres types de codages qui semblent plus naturels dans le cadre d'une réécriture de requêtes (remplacement par expressions régulières, ou remplacement par systèmes relationnels, dans le style de Prolog) s'adaptent mal aux contraintes évolutionnaires. Il est difficile, par exemple, de définir des mutations non destructrices sur une expression régulière, dans laquelle chaque élément conditionne la pertinence de l'ensemble de la partie d'expression qui le suit. Définir des opérateurs adaptés à cette structure impose de revenir à une représentation alternative sous forme d'automates, qui peut dans certains cas être codée sous forme de DAG...

Malgré l'absence de spécificité des codages en arbres au problème de la réécriture, ils apparaissent donc comme une base de travail incontournable. Si ce sont ici les contraintes du paradigme évolutionnaire qui nous imposent cette solution, encore faut-il pouvoir l'adapter à nos besoins, tant en termes de capacité de description (permet-il de définir des modèles de réécriture assez complexes) que de performance.

Automates, complexité

Nous souhaitons pouvoir traiter des requêtes, qui dans leur plus grande généralité sont des expressions booléennes. Une requête ne comportant que des termes peut en effet être envisagée comme une requête booléenne en insérant entre deux termes un opérateur implicite ("vide")[5].

Une telle expression est de fait un arbre, dont les noeuds sont des opérateurs unaires ou binaires, et les feuilles des termes. Il n'existe ni limite logicielle ni maximum pratique à la profondeur d'un tel arbre[6], ce qui signifie que nous devons être capables de traiter des structures arbitrairement complexes.

Si nous n'avons pu utiliser un codage par expression régulière, envisager les contraintes pesant sur une représentation alternative par automates est un bon moyen d'étudier notre problème sur le plan algorithmique. Pour nous placer sur ce plan, envisageons une requête valide comme un mot, construit sur un alphabet bien défini, puisque construit comme l'ensemble des termes disponibles dans l'index accompagné des opérateurs booléens. Le paragraphe précédent se

[5] C'est d'ailleurs ainsi que les traitent les moteurs de recherche, en utilisant un "*et* faible" comme opérateur implicite.

[6] Comme le montre l'examen des historiques de recherche d'Ulix.

traduit alors comme la nécessité de pouvoir identifier des mots de longueur arbitrairement grande. Cela est impossible avec des automates finis. Seuls des automates à mots infinis, ou automates de Rabin-Miller, peuvent nous apporter le niveau de généralité nécessaire (voir [MNa66, MNaPap71, MNa74]).

Si les automates à mots finis sont traduisibles sous forme d'arbres (il suffit pour cela de développer les états multiples), tel n'est pas le cas des automates de Rabin-Miller. Les développer sous forme d'arbre demanderait l'introduction d'une autre forme d'évaluation de ceux-ci. La forme classique repose sur l'application de chaque opérateur noeud à ses enfants, puis le remplacement du sous-arbre dont il est la racine par ce résultat, en procédant des feuilles vers la racine. Le développement d'automates de Rabin-Miller demanderait de pouvoir exécuter plusieurs fois un même sous-arbre, et donc l'introduction de feuilles variables, ainsi que le renversement du mode d'exécution : de la racine vers les feuilles. Cette méthode est réalisable (le modèle décrit section 2.1 s'en approche), mais peu efficace dans le cadre des arbres, et difficilement gérable sur le plan des opérateurs.

La représentation informatique classique de tels automates, sous forme de graphes d'états, n'est pas non plus adaptée à la définition d'opérateurs génétiques. Il nous faut trouver une représentation alternative, une généralisation des arbres fournissant l'expressivité des automates de Rabin. Un langage très utilisé en intelligence artificielle, le Lisp, nous fournit une base de réflexion.

Langages concaténatifs / langages à pile

Lisp est un langage très utilisé en IA, essentiellement parce qu'il permet de manipuler de la même manière langage et données. Ce n'est pas cette propriété qui nous intéresse ici, mais la similitude que présentent le codage d'un arbre dans ce langage, et celui d'un sous-programme. Une formule algébrique codée sous forme d'arbre en Lisp ne se distingue du sous-programme effectuant la même opération que par l'instruction **quote**... permettant de ne pas l'évaluer. Pourtant, la capacité de description de Lisp englobe les automates de Rabin-Miller, grâce à la présence d'une instruction de bouclage, permettant de réutiliser (ré-exécuter) plusieurs fois le même sous-programme.

Le Lisp permet aussi de définir aisément des opérateurs génétiques, comme ceux employés sur des arbres. Cela provient du codage de ces arbres sous formes de listes, dont des éléments (atomes ou sous-listes) peuvent être modifiés, supprimés ou ajoutés en conservant une structure valide. Une variante de Lisp, le RPL, fait apparaître ce fait encore plus clairement. Codant une instruction sous

la forme [arg-N ... arg-1 instruction], il supprime la notion de sous-programme directement exécuté (liste sans **quote**) et fait appel explicitement à la pile, où sont déposés et repris les arguments. Les opérateurs génétiques classiques opérant sur des arbres (interversion, suppression, ajout) se codent alors de manière identique à ceux que l'on peut définir sur des chaînes d'atomes de longueur variable, ou même des chaînes binaires.

De manière générale, un langage à pile, possédant une instruction permettant de répéter un nombre de fois arbitraire le même sous-programme, répond à nos attentes. Un langage concaténatif, dans lequel toute sous-chaîne est un sous-programme valide, permet de définir aisément les opérateurs génétiques. Cela amène à un langage peu structuré, exécuté linéairement (atome après atome), basé sur une pile d'arguments pouvant contenir données et sous-programmes. Cette représentation étant assez proche du mode d'exécution de programmes assembleur, il est en outre possible de concevoir des interpréteurs rapides.

La technique d'"Evolution Grammaticale" décrite au Chapitre 2 (section 4.5) nous aurait permis de contourner le problème, en écrivant directement les programmes résultats en C. Cependant, elle ne nous épargne pas de formaliser la structure de programmes que nous souhaitons adopter (i.e. le niveau de complexité que nous leurs permettons d'atteindre), ni de créer les outils de transformations spécifiques du problème traîté (réécriture de requêtes). En outre, elle possède un inconvénient important pour la maîtrise à long terme du système : l'absence de lisibilité. Il est en effet impossible de comprendre les individus (chaînes binaires, ou gros programme C après transcription) évolués de cette manière.

2.2 OKit : une boite à outils GP

La nécessité de structures de codages des génomes plus puissantes et plus malléables en programmation génétique n'est pas limitée au cas que nous explorons. Et, comme souvent dans le contexte de l'évolution artificielle, les outils développés pour un problème spécifique possèdent un domaine d'applicabilité beaucoup plus large. Il est en outre fréquent que la généralisation de ces outils soit de fait bénéfique pour le problème qui a été à l'origine de leur développement. C'est dans cet esprit que nous avons cherché à formaliser de manière générale les caractéristiques du langage de codage des individus dans ELISE, sous la forme d'une boite à outils pour la programmation génétique : **OKit** [@OKit].

Structure du langage

OKit est un langage typé, comportant trois types atomiques (**nombre**, **texte** et **instruction**) et un type composite **liste** pouvant contenir un nombre quelconque d'objets de chacun des quatre types précités, comme décrit figure 3.2.

Objets			
	Nombre	[scal]	3.1415926e+00
	Chaîne de caractères	[scal]	"texte quelconque"
	Liste	[list]	[obj1 obj2 ...]
	Instruction	[exec]	<NOM> ou <NOM :données>

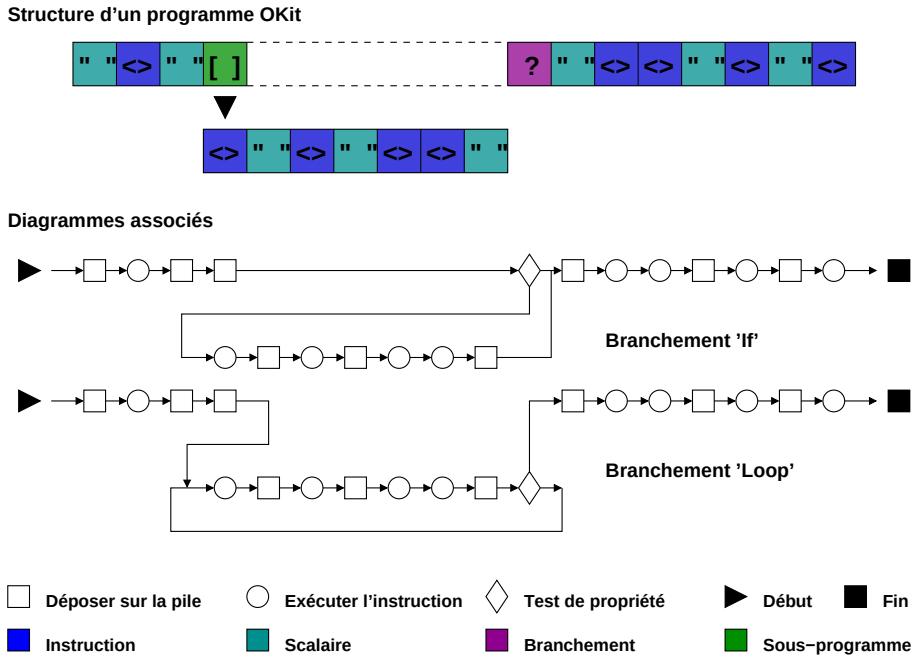
FIG. 3.2 – OKit : types et syntaxe

Un programme OKit est une liste, qui peut par exemple prendre la forme [7] :

```
[
  "texte 1" 2 <INST-1:[3 4]> "texte 2" "texte 3"
  [ <INST-2> "chaine" <INST-3> ]
  <INST-4>
]
```

Le contexte d'exécution d'un tel programme est constitué d'une pile sur laquelle sont présents les arguments du programme. Une table de variables, permettant de stocker des paramètres, n'est pas utilisée dans le cadre d'ELISE. L'exécution proprement dite consiste à examiner chacun à leur tour les éléments de cette liste, et à les déposer inchangés sur la pile, s'il s'agit d'un type scalaire (indiqué par le signet [scal] dans le tableau 3.2 : nombre ou texte) ou d'une liste, ou à les exécuter s'il s'agit d'une instruction. Chaque instruction prend ses arguments et retourne ses résultats sur la pile. Un sous-programme n'est pas différent d'une liste de données, et n'est exécuté que s'il est suivi d'une instruction dont le rôle est précisément d'exécuter ses arguments, par exemple une instruction de branchement conditionnel, ou une instruction de répétition, comme dans la figure 3.3. Ainsi, la liste [<INST-2> "chaîne" <INST-3>] de l'exemple précédent est tout d'abord déposée sur la pile, et n'est exécutée que si tel est le rôle de <INST-4>.

[7] La présentation sur plusieurs lignes n'est pas impérative, elle a été adoptée pour faciliter la lecture.



Structure d'un programme OKit, comportant un sous programme (**""** indique une chaîne, <> une instruction et **[]** une sous-liste). L'instruction suivant immédiatement le sous-programme est une instruction de branchement, prenant en argument une liste (le sous-programme). Son exécution se traduit par une diversion du corps principal du programme dans le sous-programme. On sort de la diversion en reprenant l'exécution principale là ou on l'avait laissée.

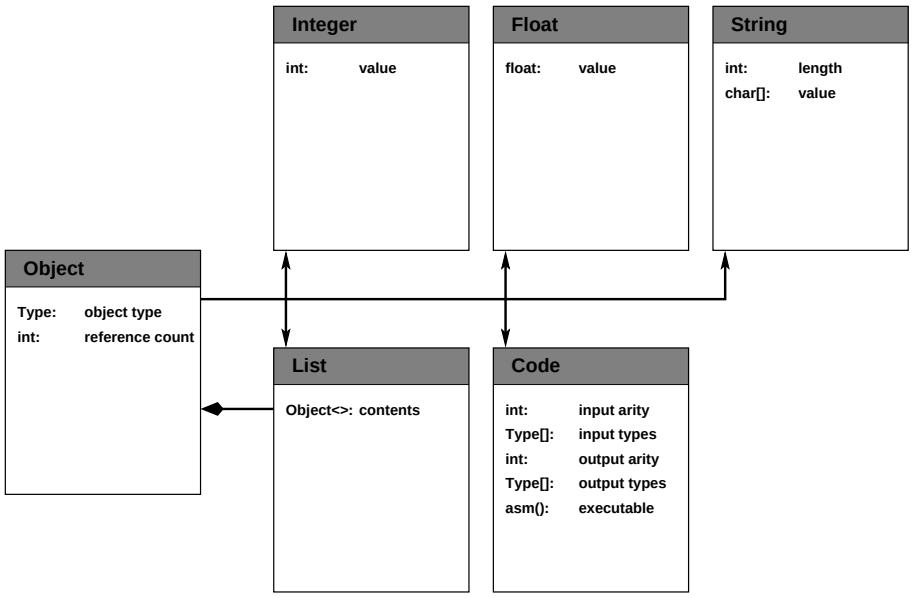
Les conditions d'entrée et de sortie de la diversion sont en revanche variables, comme le montrent les diagrammes d'exécution : s'il s'agit d'une instruction de branchement, du type *if*, la diversion n'a lieu que si un test est validé. S'il s'agit d'une instruction de répétition, du type *do... while*, la diversion est répétée au début tant qu'un test échoue.

FIG. 3.3 – OKit : structure et exécution.

Structures informatiques

L'implémentation pratique d'un tel langage nécessite l'emploi de plusieurs modèles algorithmiques particuliers, permettant de répondre en même temps aux contraintes énoncées plus haut : exécution rapide, suivi des erreurs, et manipulation aisée des structures par une interface de programmation (API, nécessaire à la définition des opérateurs génétiques).

Structure objet hiérarchique. Les opérations de pile (ajout et retrait d'arguments) agissent de la même manière sur les deux types de scalaires, les listes et les instructions. Les opérations d'exécution en ligne (programme principal)



La hiérarchie des types dans OKit a pour racine un type **Objet** générique abstrait (dont les méthodes ne sont que virtuelles : elles définissent un mode d'appel, mais pas l'effet de cet appel). Les types dérivés **Entier**, **Réel**, **Texte**, **Liste** et **Instruction** implémentent ce type abstrait en définissant l'effet de l'appel des méthodes. Dans ce diagramme UML simplifié, ces méthodes n'apparaissent pas en raison de leur nombre important.

FIG. 3.4 – OKit : diagramme de types

traitent de manière distincte scalaires et listes, d'une part, et instructions, d'autre part. Enfin, l'exécution différée (exécution de sous-programme) définit encore deux groupes distincts, scalaires d'une part, instruction et liste d'autre part. Gérer l'identité et les distinctions de traitement de ces types impose d'utiliser un modèle objet hiérarchique.

Un type objet générique (invisible du point de vue du langage, puisqu'il s'agit d'un type abstrait) est utilisé pour toutes les opérations de pile, et comme argument générique des instructions (celles-ci peuvent en effet accepter, ou refuser, certains types d'argument déterminés à l'exécution). Ce type abstrait est ensuite implémenté pour chaque type effectif du langage. Le type générique est utilisé pour construire le type liste, celui-ci pouvant contenir tout type d'objet (figure 3.4).

Gestion de la mémoire. La gestion "automatique" de la mémoire offre une plus grande flexibilité pour une application à la PG, où les programmes sont gé-

nérés automatiquement. Gérer explicitement la mémoire impliquerait en effet une phase de vérification (voire de réécriture) des programmes pour s'assurer de leur bonne exécution. Cette tâche n'est en outre pas toujours possible avant l'exécution effective du programme, lorsque son comportement n'est pas déterministe, rendant alors indispensable l'implémentation du "garbage collecting".

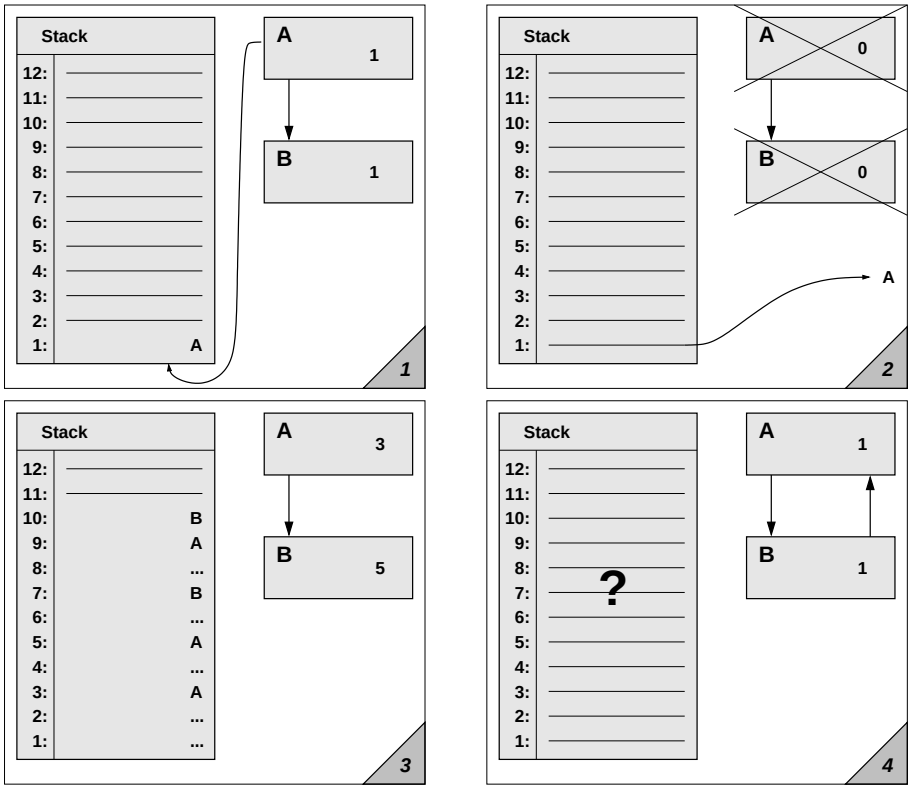
Deux techniques permettent de supprimer automatiquement les objets qui ne sont plus utilisés : le marquage, et le comptage de références. La seconde est plus rapide et conduit à une utilisation de la mémoire plus réduite que la première, mais possède le défaut d'être incapable de libérer des structures se référençant elles-mêmes, directement ou récursivement.

Le comptage de références se base sur une seule information, contenue dans chaque objet : le nombre d'objets ayant besoin de celui-ci pour exister, directement ou indirectement. Autrement dit, le *compte de références* d'un objet est le nombre d'autres objets "pointant" vers lui.

Si un objet A référence un objet B, le compte de références de B est incrémenté chaque fois que celui de A l'est, par exemple lorsqu'il est dupliqué sur la pile, ou présent une fois supplémentaire dans une liste. Lorsque A n'est plus utilisé, son compte de références est décrémenté, et tombe à 0 lorsque aucun autre objet ou niveau de la pile ne le référence. Il peut alors être supprimé de la mémoire. Si B n'était pas utilisé ailleurs, son compte de références est lui-aussi tombé à 0 (puisqu'initialisé à 1 en même temps que A) et B est supprimé. Si ce n'était pas le cas (B a par exemple été extrait d'une liste A pour apparaître sur la pile, opération pendant laquelle son compte de références a été incrémenté), B est conservé (figure 3.5).

Supposons maintenant que B fasse à son tour référence à A : A étant toujours utilisé par au moins un objet, son compte de références ne peut tomber à 0. Il en est de même pour B, et le couple A-B ne peut être purgé, même lorsqu'il n'a plus d'utilisation ailleurs. Ce défaut du comptage de référence ne se produit que lorsque des références cycliques sont possibles. La structure de langage que nous avons adoptée, complétée par une programmation adéquate des instructions, interdit l'apparition de tels cycles lors de la compilation d'un élément de code. Pour conserver cette propriété du résultat de compilation tout au long de l'exécution, les instructions doivent **renouveler** les objets qu'elles manipulent (en faire une copie indépendante en mémoire lorsqu'il s'agit de listes dont le compte de référence est supérieur à 1).

Ensemble d'instructions. Une partie importante de l'efficacité et de l'utilisabilité du langage est liée à l'ensemble des instructions disponibles. Elles défi-



Examinons un cas de référencement simple : un couple d'objets, présentant une dépendance directe (A dépend de B). Lorsque A posé sur la pile, scène *1*, le compte de références des deux objets est incrémenté. Lorsque A est retiré de la pile (scène 2), ce compte est décrémenté, et passe donc à zéro. C'est le signal qui entraîne la destruction des deux objets A et B, qui ne sont plus utilisés. A et B peuvent être traités séparément, dans ce cas les comptes de références peuvent différer. Cependant, celui de B est toujours plus grand que celui de A, comme on le voit scène *3*, où A puis B seul ont été posés plusieurs fois sur la pile. A est présent 3 fois, d'où son compte de références égal à 3. B est présent 2 fois, d'où un compte de référence égal à 5 = 3+2. Le cas problématique des références croisées, interdit dans OKit, est illustré scène *4*. Nous avons procédé comme en *1*, mais lorsque nous retirons A de la pile, il est impossible de décrémenter les comptes de références : tant que A existe, il a besoin de B, et vice-versa. La mémoire ne peut alors être vidée.

FIG. 3.5 – OKit : comptage de références

nissent entièrement les actions réalisées par un programme. En outre, ce langage n'est équivalent aux automates de Rabin-Miller que lorsque sont définies des instructions permettant l'exécution multiple de sous-programmes. Une telle instruction "<**LOOP**>" pourrait par exemple prendre une liste sur la pile (dont l'exécution se terminerait obligatoirement par le dépôt d'un nombre sur la pile) et l'exécuter tant qu'elle ne produit pas le nombre 0.

Bien sûr, l'ensemble d'instructions choisi dépend essentiellement du problème à traiter, et n'est pas du ressort d'un outil générique comme OKit. L'ensemble d'instructions choisi pour ELISE en particulier fera l'objet plus loin d'une discussion étendue. Il est en revanche du ressort du langage et des outils logiciels développés pour OKit de faciliter leur définition. En particulier, assurer l'exécution correcte des programmes nécessite d'associer d'autres informations qu'une simple action (en fait, un morceau de code machine) à une instruction.

Certaines instructions (comme **LOOP** défini plus haut) n'ont de sens que sur certains types de données. Lorsque l'exécution fournit à une instruction un type de données invalide, il est possible soit d'ignorer silencieusement l'erreur, en fournissant un résultat par défaut, soit d'arrêter immédiatement l'exécution du programme. Quelle que soit la solution retenue, il est souhaitable, dans le cadre d'une application à la PG, d'être en mesure d'identifier les programmes invalides et les causes d'invalidité. Cela permet en particulier de diriger l'application des opérateurs génétiques pour maîtriser la proportion d'individus invalides. Pour en faciliter la gestion, ce contrôle de types et de nombre des arguments a été intégré à la chaîne d'exécution. Un type invalide ou un nombre d'arguments trop faible déclenche une erreur, indiquant l'instruction concernée et le type des arguments invalides.

Cela implique de conserver ces informations au sein d'un objet **Instruction**. Si l'exécution de certains programmes peut échouer, du point de vue évolutionnaire nous sommes en présence d'individus infaisables. Réaliser une "réparation" des individus, ou vérifier au préalable leur faisabilité, fait partie des opérations que nous souhaitons pouvoir réaliser, et qui seraient grandement facilitées par des informations sur les objets retournés par une instruction.

Nous avons finalement adopté une solution composée de deux blocs, spécifiant le nombre des objets et la liste des types possibles, que nous conservons pour chaque instruction : l'un en entrée, l'autre en sortie.

Instructions standard. Quelle qu'en soit l'application, un tel langage nécessite le plus souvent la définition d'instructions de base permettant de manipuler la pile (rappelons qu'il s'agit de la seule "mémoire" d'objets dont nous disposons).

Les plus classiques sont :

- <**DUP**> : duplique le premier objet de la pile

```
4:  .......        4:  .......
3:  .......        3:    obj-2
2:    obj-2   =>   2:    obj-1
1:    obj-1        1:    obj-1
```

- <**DROP**> : supprime le premier objet de la pile

```
4:  .......        4:  .......
3:  .......        3:  .......
2:    obj-2   =>   2:  .......
1:    obj-1        1:    obj-2
```

- <**SWAP**> : intervertit les deux premiers objets de la pile

```
4:  .......        4:  .......
3:  .......        3:  .......
2:    obj-2   =>   2:    obj-1
1:    obj-1        1:    obj-2
```

- <**ROLLDN3**> : amène au sommet de la pile l'objet au niveau 3

```
4:  .......        4:  .......
3:    obj-3        3:    obj-2
2:    obj-2   =>   2:    obj-1
1:    obj-1        1:    obj-3
```

- <**ROLLUP3**> : amène au niveau 3 l'objet au sommet de la pile

```
4:  .......        4:  .......
3:    obj-3        3:    obj-1
2:    obj-2   =>   2:    obj-3
1:    obj-1        1:    obj-2
```

C'est l'opération inverse de <**ROLLDN3**>. On peut aussi définir <**ROLLDN4**> et <**ROLLUP4**>, etc...

Interface, utilisation et efficacité

Destinés au cadre de la PG, les éléments du langage OKit sont appelés à être manipulées exclusivement via des programmes, et non par le programmeur comme dans la plupart des langages classiques. Il est donc essentiel de disposer d'une interface de programmation (API) adaptée, permettant un codage aisé des instructions et des opérateurs génétiques.

Cette API travaille sur la forme binaire des objets (représentation interne obtenue à partir du "code source"). Elle permet tout d'abord de transformer un objet de sa forme texte en forme binaire et inversement. Elle fournit aussi les

utilitaires essentiels pour manipuler chaînes et listes (insertion / découpage par blocs), et vérifier les arguments d'une instruction. Elle apporte aussi les méthodes permettant une manipulation efficace de la pile (cruciale pour le maintien des performances en exécution). Surtout, elle gère l'exécution des programmes, de manière à obtenir des performances compatibles avec le contexte industriel dans lequel se place le prototype ELISE (rappelons qu'un tel programme est en réalité un individu d'une population pouvant en compter un nombre important, et sera évalué à chaque génération).

L'exécution d'un programme (une liste, dans la hiérarchie objet de OKit) sous sa forme textuelle se déroule en deux temps : une première étape de compilation, pour obtenir une forme binaire standard (sorte de "bytecode", distinct du code natif propre au matériel), puis l'exécution de ce binaire par une "machine virtuelle". Disposer d'une telle structure pré-compilée permet en particulier d'accélérer le traitement des boucles et des sous-programmes, et l'appel des instructions. L'exécution s'effectue en utilisant la pile (ses arguments sont donc les objets présents sur la pile avant l'exécution, et elle y renvoie ses résultats). Elle peut soit réussir, soit échouer. Dans ce cas, on obtient un code d'erreur, indiquant la portion de code responsable et les raisons de l'erreur.

Lorsque le programme résulte d'une construction ou d'une manipulation via l'API, il est déjà sous sa forme binaire. La forme textuelle ne sert donc qu'au stockage à long terme du code. Mais nous verrons qu'il est important de pouvoir réaliser ces sauvegardes pour examiner les populations, et même indispensable dans le cas de profils utilisateurs stockés sur disque entre deux générations.

L'essentiel du coût de l'évaluation d'un programme est généré par la machine virtuelle, même si le coût de compilation est proportionnellement plus important pour les programmes de petite taille. Il est difficile de fournir une évaluation absolue de sa performance, celle-ci dépend très fortement de la complexité des instructions définies, et de l'efficacité avec laquelle est codé le programme testé. Cependant, quelques tests réalisés sur des tâches numériques (factorisation de nombres, voir annexe 3), avec un ensemble d'instructions minimal (opérations arithmétiques et manipulation de la pile), donnent une "échelle" de 10000 (10000 pas de temps processeur pour un atome du programme). Si ce chiffre n'a pas de valeur de comparaison avec un autre langage, il fournit en revanche une indication rapide du temps nécessaire à l'évaluation des individus d'une population de taille fixée, comportant un nombre d'atomes moyen connu[8].

[8]Par exemple, nous avons utilisé des populations de taille 100, dont les individus comportaient en moyenne 10 atomes, ce qui fournit une évaluation de la population en 10ms sur une machine à 1GHz. Par comparaison, le temps d'obtention de la réponse à une requête peut varier de 5ms à 5000ms selon la complexité des outils de TR utilisés.

2.3 Opérateurs en GP

Les outils de sélection et les opérateurs définis en PG sont aussi divers que la signification concrète des codages adoptés, ceux-ci devant respecter la structure (la topologie, lorsque cette notion a un sens) de l'espace de recherche, souvent plus complexe que dans des algorithmes évolutionnaires à base numérique. Néanmoins, lorsqu'un codage à base d'arbres, ou de structures similaires, est adopté, ces outils sont dictés par la structure, très contrainte, sur laquelle ils doivent opérer. Ainsi, la plupart des opérateurs utilisés pratiquement sont définis grâce à un nombre restreint d'opérations élémentaires. Dans le cas d'OKit, avoir relâché les contraintes portant sur le langage, outre fournir un niveau de description plus élevé, permet de définir quelques opérations élémentaires. Il ne s'agit pas forcément d'une liberté accordée au programmeur. Il doit au contraire s'assurer de la qualité du code produit par les opérateurs qu'il définit, puisque celle-ci n'est plus entièrement garantie par la structure du langage.

Opérateurs primaires sur du code

Par sa conception, le code OKit adopte une structure proche de celles des arbres souvent utilisés en GP. Il est possible d'adapter les outils et opérateurs conçus pour les arbres. Insertions, suppressions ou échange de sous-arbres et feuilles se traduisent par les mêmes opérations sur des listes ou des scalaires. La modification d'un noeud se traduit comme la modification d'une instruction. De même que dans le cas des arbres, l'insertion ou la suppression n'est pas toujours possible, lorsque l'arité des noeuds / instructions l'interdit. Changer une liste en scalaire possède un équivalent pour les arbres, qui serait le remplacement d'un sous-arbre par une feuille. Tel n'est pas le cas du remplacement d'une instruction par un scalaire, qui reviendrait à supprimer un noeud en conservant ses enfants[9], ou de l'échange entre une instruction et une liste. Ces opérations peuvent fournir la base de mutations de la structure des programmes plus profondes que celles offertes dans le cadre des arbres, comme on le verra section 3.4.

En revanche, la vérification des types pose problème. Cette notion n'a pas d'équivalent dans le cas des arbres, puisque chaque noeud prend comme argument les résultats des arbres descendants, du même type que les feuilles. La distinction entre types scalaires n'est pas un réel problème : sans perte de généralité, il aurait été possible de se contenter d'un seul type chaîne. En revanche, il n'est pas possible de se passer du type "sous-programme", puisque nous devons

[9] Il s'agit d'une génération d'intron, technique parfois utilisée en PG, pour conserver du matériel génétique inexprimé.

pouvoir le fournir en argument d'une instruction "bouclante". Cette vérification est rendue plus difficile par le fait qu'un argument peut être le résultat de l'exécution, un nombre de fois *a-priori* impossible à déterminer, d'un sous-programme. En conséquence, il est parfois impossible de déterminer de quel type sera l'objet contenu à un certain niveau de la pile lors de l'exécution d'une instruction particulière.

Lorsqu'on change une instruction en une autre ou un sous-programme en un autre, on s'expose ainsi à transformer un programme valide en un programme dont l'exécution échoue. Les méthodes évolutionnaires disposent de nombreux moyens de gérer ce problème, et cet effet peut même être souhaitable dans certains cas (parties topologiquement isolées de l'espace de recherche, par exemple). Mais évaluer des programmes invalides, qui ne produiront donc pas de résultat, implique un coût algorithmique qu'il est nécessaire de minimiser. Pour cela, il est possible soit de limiter l'apparition de tels programmes, soit de détecter avant l'exécution, de manière beaucoup moins coûteuse, le "plantage". Dans les deux cas, il est nécessaire de disposer des outils permettant d'analyser l'évolution des types présents sur la pile au cours de l'exécution, d'identifier les types requis en entrée par une portion de code, et ceux des résultats qu'elle produit.

Outils de description de code : arité, prototypes

Le suivi des types, qu'il s'agisse d'arguments ou de résultats, repose sur l'identification des types des objets présents sur la pile à un moment donné. Nous appellerons cet ensemble de types "prototype", nous parlerons donc de prototype d'entrée et de sortie d'une section de code.

La structure des instructions OKit précise le nombre d'objets et les types possibles dans ces deux cas, fournissant alors une "superposition" de prototypes possibles. Cette notion peut être étendue à l'exécution d'autres types, en définissant comme vide le prototype d'entrée d'une liste ou d'un scalaire, et comme le prototype réduit à un seul objet de ce type leur prototype de sortie (ceux-ci sont en effet simplement déposés sur la pile lors de l'exécution). On obtient ainsi la notion de "signature" d'un objet, qui résume les modifications possibles des types des objets présents sur la pile lors de son exécution.

Pour disposer d'une notion de signature pour une portion de code, il suffit alors de définir le comportement de ces notions vis-à-vis de la concaténation, toute portion de code pouvant s'exprimer comme la concaténation de concaténations ... de portions de code ultimement réduites à un seul élément. Il nous faudra simplement vérifier, pour valider la cohérence d'une telle définition, l'as-

sociativité des opérations définies. Pour cela, nous allons formaliser la notion de signature, ou superposition de prototypes.

Un prototype est un a-uple de types, où a est l'arité. Ce qui nous permet de formaliser un prototype comme un couple $(p, t), p \in \mathbb{N}, t \in \mathcal{T}^p$. Chaque élément de code possède deux ensemble de tels couples (correspondant éventuellement chacun à des arités différentes). Nous dirons qu'un prototype (p_1, t_1) est inclus dans un prototype (p_2, t_2) si $p_1 < p_2$ et $t_1^i = t_2^i \ \forall i \in [[1, p]]$. Nous dirons qu'ils sont compatibles si l'un est inclus dans l'autre, ou l'inverse. Il apparaît immédiatement que l'exécution d'une portion de code dont un élément ne possède pas de prototype de sortie compatible avec le prototype d'entrée de l'élément qui le suit ne peut s'exécuter.

Dans le cas où il n'existe pas de telle incompatibilité, ces deux éléments possèdent au moins deux prototypes compatibles. Si le prototype de sortie est inclus dans le prototype d'entrée, nous construisons de nouveaux prototypes d'entrée de la paire d'éléments par l'ajout des types complémentaires aux prototypes d'entrée du premier élément. Prenons $(p_{1,s}, t_{1,s})$ inclus dans $(p_{2,e}, t_{2,e})$. Pour chaque $(p_{1,e}, t_{1,e})$, nous construisons (p, t) avec $p = p_{1,e} + p_{2,e} - p_{1,s}$, et $t^i = t_{1,e}^i$ si $i < p_{1,e}$, $t_{1,e}^i$ sinon. Dans le cas inverse ($(p_{2,e}, t_{2,e})$ inclus dans $(p_{1,s}, t_{1,s})$), nous réalisons la même opération en intervertissant prototypes d'entrée et de sortie.

Pour chaque paire de prototypes compatibles, nous produisons ainsi un ensemble de prototypes d'entrée ou de sortie. La réunion de ces ensembles, pour chaque paire compatible, fournit les superpositions de prototypes d'entrée et de sortie de la paire d'éléments de code. La démonstration de l'associativité, simple mais longue, a été reportée en annexe 2 (section 2.2), qui reprend par ailleurs une formalisation plus rigoureuse de ce développement.

Cette notion de signature, et les méthodes de calcul afférentes, sont aisément traduisible en algorithmes. Qui plus est, leur calcul nécessite peu d'opérations par instruction (quelques parcours de vecteurs –prototypes– de petite taille), permettant de vérifier la compatibilité de tous les éléments du code, et le calcul des arguments requis par un programme. Cependant, certaines instructions, dont les prototypes ne peuvent être définis utilement (instructions acceptant tous les types, par exemple) rendent parfois inutile ce calcul, le résultat étant une superposition de tous les prototypes possibles pour une arité donnée. Cet outil de vérification permet d'éliminer rapidement les programmes clairement défectueux et de ne pas effectuer des opérations de transformations du code introduisant des défauts, mais ne garantit pas l'exécution sans erreur de toutes les sections de code produites en y ayant recours. Le cas typique est par exemple celui d'un programme comportant une instruction de boucle.

3 Elise, flux de données et modularité

3.1 Vue d'ensemble

Rappelons-le, les deux contraintes fondamentales pesant sur le développement d'ELISE sont l'intégration aux outils de TR préexistants et le fonctionnement dans un contexte client-serveur. Nous avons vu, section 1.2, que ces contraintes imposent, respectivement, une architecture composée de modules distincts, et un mode de communication des modules reposant sur des flux de données précisément formalisés.

Certains de ces composants et leurs interfaces (interface utilisateur, recherche dans les index) ont déjà été décrits, et contraignent la structure d'ELISE. Il nous reste à préciser comment l'algorithme évolutionnaire traite les informations récoltées par le suivi du parcours de l'utilisateur, et par quel moyen les informations contenues dans le profil permettent la retranscription des requêtes.

3.2 L'algorithme GP

L'élément déterminant dans la conception de l'algorithme de programmation génétique servant à optimiser les profils est le codage des individus, autrement dit le contenu physique des profils. De là découlent les modes d'évaluation (plus exactement, de calcul d'une évaluation à partir des temps passés sur chaque document), les opérateurs génétiques d'initialisation et de reproduction, et les méthodes de paramétrage.

Première implémentation

Nous avons montré section 2 que, pour remplir la tâche de réécriture des requêtes, les codages en arbres, bien que pratiques du point de vue évolutionnaire, étaient insuffisants. Nous avons donc développé OKit pour répondre à ce besoin.

Les premières implémentations réalisées reposaient donc sur une population dont les individus étaient des programmes OKit. Un jeu d'instructions spécifique, que nous examinerons section 3.3, permet à ces programmes d'opérer sur une requête (fournie sous forme de liste sur la pile) pour produire une version transformée.

Dans cette version, à chaque génération la requête est transformée par tous les individus, fournissant autant de "requêtes images" que d'individus dans la population. Chacun de ces résultats est alors passé au moteur booléen, et les

listes de documents retournés sont agrégées et triées avant d'être proposées sous forme d'une liste unique à l'utilisateur. L'évaluation des individus se fait en totalisant, pour chacun d'eux, le temps passé par l'utilisateur sur tous les documents issus d'une requête transformée par cet individu.

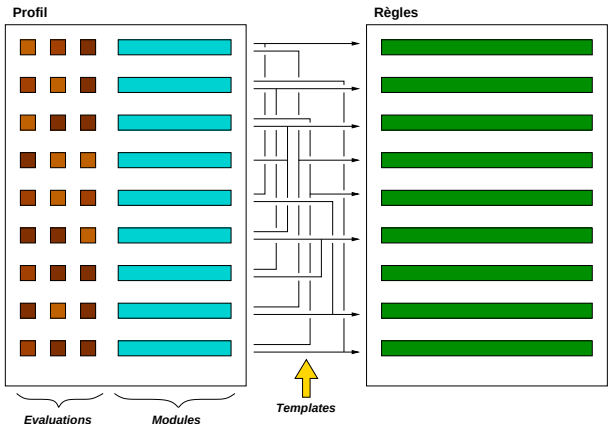
Cette méthode nécessite, pour pouvoir prendre en compte une diversité de formats et de sujets de requêtes suffisante, une taille de population élevée : pour des tailles inférieures à la centaine, une telle évaluation à chaque requête des individus ne permet pas d'apprentissage efficace. Les performances maximales des individus restent basses, et ceux-ci sont renouvelés fréquemment, ce qui ne permet pas à l'algorithme évolutionnaire de fonctionner correctement.

De manière simplifiée, on réalise alors une exploration aléatoire, sans mémoire. Il est possible d'améliorer les résultats en cumulant les évaluations de plusieurs générations pour un individu conservé (rappelons que, selon le modèle parisien, une grande partie de la population est conservée inchangée d'une génération à l'autre). Par exemple, la fitness interne serait une moyenne pondérée (3/10,3/10,2/10,1/10,1/10) des 5 dernières évaluations.

Si les résultats sont nettement améliorés lorsque l'on présente au système des requêtes simples et sur des sujets similaires, il suffit d'une succession de quelques requêtes très différentes pour voir réapparaître une saturation de la performance. De plus, des individus "parasites" (dont l'évaluation est toujours mauvaise mais jamais nulle) tirent parti de ce mode de calcul.

Augmenter nettement la taille de population n'est pas possible (chaque traitement demanderait alors trop de temps de calcul, et les listes de résultats seraient déraisonnablement volumineuses). De plus, ce mode de codage souffre d'un mal très courant en programmation génétique : l'inflation des tailles d'individus, ou "bloat" (voir les travaux de W.B. Langdon [LanPol97], et plus récemment [ArrSmi98]). Cela résulte principalement du raffinement des individus pour approcher au plus près un optimum, avec une précision largement inférieure au bruit, et donc sans valeur. Le bloat contribue au coût algorithmique d'évaluation des populations, car un individu est en moyenne d'autant plus long à évaluer qu'il est volumineux. S'il est possible de limiter le bloat en pénalisant les individus de trop grande taille, cela limite aussi la richesse d'expression des individus, et nuit à la qualité des résultats.

L'examen direct des populations offrant des performances raisonnables dans l'approche précédente (avec une évaluation sans historique, limitée à la requête courante) fait apparaître une redondance importante. Les meilleurs individus comportent des sections entières totalement identiques. De plus, celles-ci ne résultent pas forcément d'une ascendance commune (bien que cela soit le plus



Le profil utilisateur est composé d'une liste de modules, comportant chacun un historique d'évaluations, servant à calculer la fitness, et le code OKit du module. Grâce à des modèles pré-définis, ces modules sont assemblés pour former les règles permettant de transformer la requête de l'utilisateur.

FIG. 3.6 – Codage d'un profil utilisateur

fréquent), mais aussi d'évolution convergente, et ne résultent donc pas toujours de la reproduction d'un seul individu particulièrement performant. Ces sections identiques réalisent donc une tâche particulière, possédant un intérêt du point de vue de l'interprétation et l'analyse des requêtes.

De cette constatation résulte un nouveau codage, décrit figure 3.6. Un individu n'est plus un programme de réécriture complet, mais un *module*, réalisant une partie de la tâche de réécriture. Celle-ci sera menée à bien en combinant plusieurs de ces modules. L'évaluation de chaque module, selon la même logique que précédemment, se fait en totalisant les temps passés sur des documents issus de requêtes transformées en utilisant ce module. Si dans le précédent modèle, la visualisation d'un document pouvait contribuer à l'évaluation de plusieurs individus par coincidence dans les listes de documents issus des requêtes transformées, une telle multiplicité est maintenant systématique, puisque plusieurs modules interviennent dans chaque requête transformée.

Génération à partir de modèles de production

Dans ce schéma, chaque transformation résulte de la combinaison de plusieurs modules. Ni la structure du langage OKit, ni le mode de transformation des requêtes (ou le mode d'exécution des programmes) n'imposent de forme

a-priori pour cette combinaison. Nous avons choisi de réaliser cette combinaison par insertion des modules au sein de modèles pré-établis, résultant de considérations logiques sur la structure des requêtes et du langage[10]. Cette méthode permet en outre d'introduire dans l'algorithme un peu de la connaissance que nous avons du problème, en favorisant la construction de règles valides, et en rapport avec la structure des requêtes.

En effet, une requête est un arbre, et donc une structure récursive pour laquelle tout et parties peuvent être traitées de même. Aussi peut-on envisager que chaque module opère sur une partie d'une requête, si et seulement si celle-ci possède une structure particulière. Ce qui conduit à envisager une combinaison, par exemple, de deux modules, comme le résultat de l'exécution du premier sur la première sous-requête de l'opérateur racine, et du deuxième module sur la deuxième partie, et ceci uniquement dans le cas où cet opérateur racine est **#AND**. Les listes de mots sans opérateur sont traitées en faisant intervenir la notion d'opérateur implicite définie à la section 1.2.

Supposons que l'on ait défini une instruction **IFAND**, testant si l'opérateur racine d'une requête est **#AND**. Elle exécute dans ce cas, sur chacune des sous-requêtes, les programmes présents sur les deux premiers niveaux de la pile. Définissons de même une instruction **DOAND**, réunissant deux sous-requêtes via l'opérateur **#AND**. On peut alors traduire l'exemple donné en langage OKit par :

```
[ @1 @2 <IFAND> <DOAND> ]
```

où @1 et @2 représentent les deux modules à introduire. Ce modèle teste par l'instruction **IFAND** si l'opérateur racine est bien **#AND**, et dans le cas positif exécute les deux modules sur chacune des deux sous-arbres de la requête. Il réunit ensuite ces deux parties, grâce à l'instruction **DOAND**. On peut aussi envisager de changer l'opérateur racine, et cette construction est applicable à l'opérateur unaire **#NOT** avec un seul module.

A chaque génération, et pour chaque module ou paire de module, un tel modèle est choisi aléatoirement, et produit une règle de réécriture qui, appliquée à la requête de l'utilisateur, fournira une liste de documents.

Si on se limite à ce type de construction, reposant sur des tests de structure de la requête simples, le nombre de modèles disponible est important (nous en

[10] Cet aspect mériterait certainement une étude approfondie, envisageant divers modes de combinaison et diverses approches d'élaboration des modèles. Ce point n'a pas été traité pour l'instant

avons construit 49), mais assez réduit pour permettre de tester une proportion importante de ces combinaisons au cours de la "durée de vie" d'un individu. Introduire un biais dans la sélection aléatoire des modèles de construction de règles permet de favoriser certaines structures semblant plus "réalistes". Il suffit pour cela d'attribuer une probabilité de sélection à chacun des modèles définis.

Contribution sémantique pour le GP

L'approche parisienne conduit à conserver une part importante de la population d'une génération à la suivante. Dans ce cas, les opérateurs génétiques sont responsables d'une part importante de la créativité. Mais réintroduire à chaque génération une petite quantité d'individus générés *ab-nihilo* peut s'avérer nécessaire pour assurer une exploration adéquate de l'espace de recherche. En effet, les opérateurs génétiques parisiens, tels que nous les avons construits, induisent des modification modérées du code et de sa structure, et n'assurent pas une diversité suffisante et une exploration assez large pour suivre rapidement un changement de thème des requêtes.

Il est possible, grâce aux outils de vérification exposés section 2.3, de générer de manière aléatoire du code OKit valide, à condition de disposer d'une liste de termes dans laquelle pouvoir choisir (les opérateurs génétiques imposent de toute manière l'emploi d'un thesaurus, voir section 3.3 – nous pouvons donc utiliser la liste de ses entrées). Il faut pour cela fixer la proportion d'objets de chaque type, le niveau de récursion (nombre maximal d'imbrications de listes admis) et éventuellement le taux d'alternance de types, si l'on choisit un mode de génération séquentiel. Cependant, le code généré possède une valeur faible, ne tenant généralement pas compte de la structure en requête booléenne de ses arguments.

Une autre méthode de génération de ces individus est par variation sur des modèles pré-établis. Il ne s'agit cependant pas de muter des individus préexistants, mais plutôt d'utiliser les informations sémantiques récoltées par analyse des requêtes ou des documents consultés pour construire des individus selon une série de modèles génériques. Par ce biais, il est possible d'améliorer la réactivité à un changement de sujet des requêtes (la méthode parisienne répond généralement lentement à un changement du paysage de fitness).

De manière similaire à la procédure adoptée pour la combinaison de modules, nous fixons un ensemble de modèles, réalisant des opérations élémentaires sur la requête (changer un terme particulier en un autre, par exemple). Ces modèles sont utilisés pour générer des individus, en explicitant les termes "laissés en

blanc" dans les modèles. Nous utilisons pour cela des termes de la requête, ou des termes extraits des documents visités. Une analyse sémantique qui permettrait de construire un "thesaurus" de paires de termes reliés dans les documents compléterait ce processus[11] : utiliser ces paires de termes dans les modèles introduit une notion de synonymie locale, spécifique de l'utilisateur, ou d'un contexte sémantique identifié dans la requête par le modèle.

Plutôt que d'employer la méthode "lourde" d'extraction sémantique, introduire, à chaque génération, des individus utilisant des termes de la requête "lisse" notablement le comportement du système. Cela permet d'obtenir une diversité suffisante des termes inclus dans les modules, tout en restant dans les domaines sémantiques réellement significatifs pour l'utilisateur. Ce comportement de lissage est d'autant plus efficace que certains des individus introduits sont conservés plusieurs générations, et participent aux recombinaisons génétiques.

Soulignons que les termes présents dans les modules servent de marqueurs, de critères de décision en quelque sorte, pour la détermination par ces programmes du contexte sémantique de la requête ou des parties de requête sur lesquels ils opèrent. Le contenu sémantique des modules est donc essentiel à leur efficacité. La présence dans ces modules de termes qui n'apparaissent jamais dans les requêtes est inutile, et les sections de code concernées pourraient être supprimées. Plus généralement, un module ne contenant aucun terme ayant un rapport de sens avec les termes de la requête ne peut participer efficacement à la réécriture de cette requête.

Instructions : un compromis entre pouvoir descriptif et taille de l'espace

Un langage à pile comme OKit peut atteindre les capacités de traitement des automates de Rabin s'il comporte une instruction de répétition. Mais son usage implique une structure particulière du sous-programme répété. Il doit fournir, à chaque exécution, un booléen indiquant si celle-ci doit être répétée ou non. Il doit être ré-exécutable autant de fois que nécessaire, alors qu'il prend à chaque exécution des arguments sur une pile de taille finie : il faut s'assurer que le contenu de la pile n'est pas consommé avant l'arrêt de l'exécution.

De plus, une telle structure met en échec les méthodes de vérification de validité que nous avons mis en place : il est impossible de s'assurer de la compatibilité des prototypes sur un nombre d'exécutions qui ne peut être déterminer avant d'exécuter effectivement le programme. De plus, la complexité de telles structures élargit considérablement l'espace de recherche. Si cela permet des

[11] Ce mécanisme n'est pas encore implémenté dans la version actuelle d'Elise.

traitements fins des requêtes, cela impose un paramétrage entraînant des temps de calcul plus importants : une taille moyenne des individus plus élevée, pour laisser la place à cette complexité, et une taille de population plus importante, pour permettre l'exploration.

Il est possible d'atteindre une capacité de traitement compatible avec nos besoins en faisant appel à une version "bridée" de l'instruction de répétition, en simplifiant le critère d'arrêt, et en assurant au préalable la présence des arguments nécessaires sur la pile. Pratiquement, cela se traduit par une instruction "<MAP>", qui exécute un sous-programme sur chaque élément d'une liste. Le sous-programme est donc exécuté exactement autant de fois que la liste compte d'éléments, et à chaque exécution l'élément concerné est présent au premier niveau de la pile.

Cette instruction est particulièrement adaptée au traitement de requêtes, structurées comme des listes de mots et opérateurs. A condition de tester s'il s'agit d'un mot ou d'un opérateur, et de laisser inchangés les opérateurs, il est ainsi possible d'exécuter répétitivement un traitement donné sur tous les éléments d'une requête. Etant donné la forme particulière des données à traiter, cette instruction apporte au langage la puissance de traitement dont nous avons besoin, sans introduire un niveau de complexité pénalisant le fonctionnement de l'algorithme évolutionnaire.

3.3 Ensemble d'instructions

La "boite à outils" OKit, qui définit une hiérarchie objet et un langage de programmation, n'est pas utilisable sans la définition d'un ensemble d'instructions, éléments fournissant les méthodes effectives de traitement des données. Aux instructions "de structure" que nous venons de décrire, permettant de gérer le flux d'exécution ou le stockage d'objets, s'ajoute un ensemble d'instructions opérant sur les données à traiter, et spécifique du problème abordé.

Réécriture d'arbres booléens

Les données que nous avons à traiter ici sont des requêtes, que l'on peut voir – et coder – sous forme d'arbres booléens. La plus grande partie des opérations de structure sur des arbres peut être construite à partir d'opérations locales (faisant intervenir un noeud et les noeuds racines de ses descendants). Les opérations plus complexes peuvent alors s'écrire comme l'application récursive de plusieurs opérations locales. Le codage d'arbres sous forme de listes imbriquées,

```
8:      ...........
7: arbre_petit_fils_2.2
6: arbre_petit_fils_2.1
5:          op_fils_2
4: arbre_petit_fils_1.2
3: arbre_petit_fils_1.1
2:          op_fils_1
1:        op_racine
```

FIG. 3.7 – Forme décomposée sur la pile d'un arbre binaire.

et l'utilisation de la pile, permet en outre de traduire ces opérations locales en déplacements d'objets sur la pile. En effet, "atomiser" partiellement sur la pile un arbre, de manière à rendre accessible les noeuds qui nous intéressent, conduit à une pile ayant une forme caractéristique. Un cas d'arbre binaire est par exemple décrit figure 3.7.

A partir d'opérations élémentaires sur la pile, de nombreuses opérations élémentaires sur les arbres peuvent être exprimées en deux instructions (parmi les instructions <**ROLLUP***k*> et <**ROLLDN***k*> définies section 2.2[12]) :

- déplacement de noeuds (ex : échanger les noeuds enfants 1 et 2)
- déplacement de sous-arbres (échanger le deuxième enfant du premier fils avec le premier enfant du deuxième fils, etc...)
- remontée ou descente de sous-arbres

De plus, toute paire d'instructions <**ROLLDN**> - <**ROLLUP**> réalise une opération sur l'arbre, décomposée sur la pile comme à la figure 3.7.

Pour pouvoir tirer parti de l'efficacité de ce mécanisme (simple, et rapide), nous avons besoin de deux instructions supplémentaires, <**SPLIT**> et <**GROUP**>, réalisant l'éclatement sous forme (racine, enfant-1,...enfant-n) et la reconstruction d'un arbre. Dans le cas d'arbres binaires, la forme ci-dessus peut s'obtenir alors à partir d'un arbre par la séquence d'instructions <**SPLIT**> <**ROLLUP3**> <**SWAP**> <**SPLIT**> <**ROLLDN3**> <**SPLIT**> <**ROLLDN7**>.

En réalité, nous avons des noeuds d'arité différentes (1, pour #**NOT**, 2 pour les autres) : il ne s'agit pas toujours d'arbres binaires. Si définir la séquence d'instruction à utiliser pour "atomiser" un arbre d'arité donnée est simple, le problème est ici que nous ne disposons pas de cette information.

[12]Rappelons que le rôle de des instructions est de ramener le niveau k au niveau 1, ou amener le niveau 1 au niveau k, pour chaque niveau k allant de 2 à 7

De plus, la séquence de 7 instructions à utiliser est aisément détruite par les opérateurs génétiques. Nous devrions donc disposer d'une instruction <**SPLIT2**> réalisant l'éclatement jusqu'aux petits-enfants, directement dans la forme ci-dessus. Elle doit bien sûr, pour les mêmes raisons, s'accompagner d'une instruction <**GROUP2**>.

Faire usage du contexte des termes de la requête et de sa structure pour effectuer une transformation adaptée au contexte nécessite de prendre des décisions (en d'autres termes, d'effectuer des branchements dans le flot d'exécution) en fonction de la structure de la requête. Cette capacité est fournie par des opérateurs de branchements conditionnés par des tests à valider sur un arbre. Il s'agit par exemple d'une instruction exécutant un sous programme sur les fils d'un arbre dont la racine est **#AND**, que nous baptiserons <**IFAND**>.

Cette instruction s'utilise sous la forme :

[[sous-programme **]** <**IFAND**> **]**

et laisse inchangée un arbre dont la racine n'est pas l'opérateur **#AND**. Dans le cas contraire (un arbre

["#AND" [fils-1 **] [** fils-2 **]]**

), la séquence précédente revient à exécuter

[sous-programme **]**

sur la pile :

```
3:      .......
2:   [ fils-2 ]
1:   [ fils-1 ]
```

On peut définir de même <**IFOR**>, <**IFNOT**>. L'opérateur implicite est détecté par une instruction <**IFVOID**>.

De fait, l'expérimentation (voir chapitre 4) montre qu'une telle construction est en réalité d'un faible intérêt pratique, en raison de la fragilité de ces structures, et de la possibilité de grouper plusieurs traitements simples réalisés par divers individus, en lieu et place d'une structure complexe.

Expansion sémantique

Les performances des solutions de TR récentes sont conçues pour offrir à l'intégrateur de nombreux points de contrôle permettant d'adapter finement les outils qui composent ces solutions aux spécificités des bases exploitées. L'un des principaux points de contrôle est fourni par des outils d'examen et d'analyse

sémantique particularisés, permettant d'atteindre des performances de l'ordre de 80%, tant en taux de rappel qu'en précision.

Ces deux critères mesurent mal le type de "performances" que nous souhaitons atteindre ici, mais n'en demeurent pas moins des indices importants, et nous ne saurions approcher les résultats que nous venons de citer sans l'utilisation de l'expansion sémantique au sein des procédures de réécriture. A la différence que, pour rester dans l'esprit de l'approche, il ne s'agit pas ici de réaliser cette expansion de manière systématique, voir brutale. Nous risquerions alors de détruire toute individualité de traitement et d'interprétation introduite par l'évolution. C'est en fin de compte au mécanisme évolutionnaire que doit revenir le contrôle de cette expansion.

Le procédé de choix pour réaliser cet objectif est bien sur d'introduire de nouvelles instructions, permettant d'effectuer une expansion "à la carte" : terme par terme, ou sous-arbre par sous-arbre, au sein d'une requête. Nous associons ainsi à chaque mode d'interrogation principal des thesauri une instruction permettant de réaliser l'expansion d'un terme. La généralisation étant aisée, ces instructions agissent en fait aussi sur des portions de requêtes, en conservant structure et opérateurs booléens inchangés[13], et en remplaçant les autres termes par la liste des termes associés selon le mode d'interrogation choisi.

Nous obtenons ainsi (pour les thesauri "classiques") quatre instructions :
- <**VOCSYN**>, réalisant une expansion synonymique,
- <**VOCGEN**>, réalisant une expansion hypéronymique,
- <**VOCSPE**>, réalisant une expansion hyponymique,
- optionnellement <**VOCANT**>, réalisant une expansion antonymique (cette relation n'est pas disponible dans tous les thesauri).

Ainsi que nous l'avons dit, ces instructions agissent indifféremment sur un terme seul (une chaîne OKit), ou une section de requête (une liste).

3.4 Opérateurs génétiques

Il existe des similitudes profondes entre les structures sur lesquelles opèrent les instructions et celles sur lesquelles travaillent les opérateurs génétiques : requêtes (arbres booléens) pour les premiers, langage OKit (par construction proche des arbres) pour les seconds. Et dans les deux cas, les transformations sémantiques et les réorganisations de structure jouent un rôle important.

Cependant, si nous avons relâché les contraintes syntaxiques et structurelles

[13] Cela explique que nous ayons choisi, par exemple, la notation **#AND** et non **AND**, pour éviter toute confusion avec un terme présent dans les thesauri.

sur OKit pour en augmenter les capacités, cela doit se traduire par une attention accrue sur la capacité des opérateurs génétiques à produire du code valide. Nous avons déjà indiqué que le paradigme évolutionnaire s'accompagnait de nombreux moyens de contrôler ou de tirer parti des individus invalides, à condition de conserver une proportion basse d'individus défectueux. Aussi les opérateurs agissant sur la structure du code, et par extension sur les instructions, dont l'arité conditionne la validité de ces structures, doivent-ils être conçus en conséquence.

Dans ce but, nous divisons les opérateurs classiques de mutation et de crossover en plusieurs classes, selon qu'ils modifient ou non la structure du code. Cela nous conduit à affecter des probabilités distinctes à ceux que nous appellerons "opérateurs locaux", qui sont sans incidence sur la structure du code, et les "opérateurs globaux", à même de modifier cette structure. Les derniers seront éventuellement accompagnés d' "opérateurs de réparation", permettant de restaurer la validité de portions de code rendues invalides.

Outils de transformation isomorphe : opérateurs locaux

Nous avons examiné section 2.3 une méthode permettant de vérifier la validité de code OKit, dans l'hypothèse où chaque instruction accepte toutes les instances de ses prototypes. Changer une instruction en une autre, ou un scalaire en un autre, en conservant prototypes d'entrée et de sortie, ne change rien aux calculs de prototypes effectués dans le cadre de cette méthode, et donc à la vérification de validité qu'elle procure. Ainsi, changer un élément de code en un autre, dans la même classe de profils d'entrée et de sortie, transforme une portion de code valide en une autre portion valide.

Cette invariance est mise à profit dans une mutation "locale", qui transforme un objet d'un type en un objet de même type, et en outre ne remplace une instruction que par une autre possédant les mêmes prototypes d'entrée et de sortie. Les résultats développés dans l'annexe 1 nous assurent alors la conservation de la validité des segments de code transformés.

Mutation des instructions

Il existe souvent peu d'instructions de prototypes équivalents. C'est le cas par exemple de certaines instructions de pile évoquées plus haut, et surtout des instructions sémantiques. Un exemple type de mutation d'une instruction est ainsi le remplacement de <**VOCSYN**> par <**VOCGEN**>. Le rôle de cette mutation est d'élargir le domaine de la transformation sémantique réalisée.

Mutation des listes

La mutation des listes ne rentre pas dans le cadre de mutations locales. Si les listes sont le plus souvent posées sur la pile tel quel, les instructions <**LOOP**> ou <**MAP**> *exécutent* cette liste, faisant alors intervenir le prototype résultant. Si cette liste était mutée en une autre, il ne s'agirait d'une mutation locale que dans le cas où les prototypes résultants coincideraient, ce qui se produirait très rarement. Et même si le prototype résultant restait inchangé, les outils d'analyse dont nous disposons ne fonctionnent que dans le cas où les prototypes définissent entièrement les cas d'échec de l'exécution. Nous réalisons donc la mutation des listes au sein des mutations globales.

Mutation des chaînes

La mutation de chaînes est plus intéressante du point de vue évolutionnaire, parce que source de points de contrôle et de paramétrage variés. Forcément basée sur l'utilisation de thesauri, elle consiste à remplacer un terme par un autre terme lié au premier dans le thesaurus, selon une des relations de synonymie, hypéronymie, hyponymie, etc... Parmi la liste de termes renvoyée par l'interrogation des thesauri, l'un deux est choisi au hasard, et remplace le terme d'origine.

Cela nous donne donc la possibilité de muter chaque chaîne de différentes manières. Décider, de manière logique et systématique, quelle transformation utiliser nécessiterait la connaissance d'un contexte sémantique du terme, bien évidemment absent dans du code. Nous réalisons donc le choix de la transformation de manière stochastique, démultipliant ainsi l'opérateur de mutation locale en plusieurs opérateurs possédant des probabilité distinctes, pour chacun des modes d'interrogation des thesauri.

Ainsi, en choisissant la relation de synonymie (la plus logique, dans de nombreux cas, et à laquelle on affectera probablement la probabilité la plus élevée), le terme "sucre" pourra être transformé en "carbohydrate" ou "saccharide", ce qui a pour effet de remplacer un terme vernaculaire en son équivalent technique, un moyen par exemple de détecter les documents contenant une analyse biochimique. Ce terme aurait aussi pu être transformé en "composé organique", par hypéronymie, ou en "ribose", "glucose", etc... par hyponymie.

Lorsque nous disposons de thesauri "flous" (rappelons qu'il s'agit de thesauri pour lesquels chaque relation entre termes est accompagnée d'une indication d'intensité), il est de plus nécessaire de fixer un "niveau" d'expansion : le seuil d'intensité de la relation à partir duquel nous allons prendre en compte un terme relié dans la liste servant au choix aléatoire. Ce seuil doit être déterminé relation par relation, car nous ne disposons d'aucune indication permettant de "normaliser" les intensités indiquées pour chaque type de relation.

Nous évoquerons plus en détails au chapitre suivant les différents aspects du paramétrage induit par ces nombreux seuils et probabilités. Si disposer de cette manière de nombreux "points de contrôle" sur l'algorithme permet de l'adapter à des situations variées, il n'en reste pas moins que cela complique singulièrement l'expérimentation.

Crossover local

Le crossover, dans le cas de chaînes binaires, de vecteurs de réels, ou d'arbres, consiste généralement à échanger entre les parents des sections assez longues du génome. Une telle forme de crossover possède aussi une signification dans le cas de listes OKit, mais change inévitablement les prototypes résultants.

Il est pourtant possible de définir une forme locale de crossover, qui consiste à échanger des éléments isolés de prototype identique entre parents. Un tel crossover n'a pas beaucoup de signification pour les instructions : vu le nombre peu important d'instructions définies, son rôle serait très proche d'une mutation.

En revanche, appliqué aux chaînes, il permet l'échange d'information sémantique entre individus. Cela permet ainsi l'intégration d'information extraites à des générations différentes dans les mêmes individus (rappelons que nous intégrons des termes provenant des requêtes et des documents récents dans les individus générés aléatoirement à chaque génération).

Opérateurs globaux

Si l'on se limite aux formes locales pour construire les descendants de la population courante, ceux-ci posséderont la même structure qu'un de leur parents. La créativité structurelle reposerait alors entièrement sur les nouveaux individus générés aléatoirement ou à partir de modèles à chaque génération. Comme ces nouveaux individus acquièrent rarement une fitness élevée, ils sont conservés peu de générations, et les éléments de structure novateurs qu'ils contiennent, ne pouvant passer à d'autres individus, sont perdus. Le seul moyen d'introduire une quantité non destructrice de créativité structurelle dans les individus déjà présents, ou d'extraire les portions intéressantes des individus créés à neuf, est d'utiliser des opérateurs de mutation et de crossover agissant sur la structure du code.

Il est possible de voir le rôle des opérateurs locaux comme une optimisation locale permettant d'améliorer du code déjà efficace, ou de suivre un optimum de fitness qui se déplace. Par contraste, les opérateurs globaux servent à promouvoir l'exploration de l'ensemble de l'espace, et à acquérir de nouveaux optima. Ces

opérateurs globaux devront probablement avoir une probabilité réduite par rapport aux opérateurs locaux, de manière à ne pas détruire des individus efficaces, mais à en modifier seulement une partie.

Mutation des instructions

Les opérateurs de mutation globale réalisent le changement d'une instruction en une autre, d'arités distinctes, ou d'un objet de type quelconque en un autre. Lors d'un changement de type, le nouvel objet est recréé aléatoirement, selon les mêmes règles qui président à la création aléatoire de nouveaux individus.

Crossover global

Le crossover global consiste quant à lui à former des descendants par échange de portions complètes de code entre parents. Son fonctionnement est le même que le crossover recombinant appliqué aux chaînes de bits : des points de coupure sont choisi aléatoirement sur chacune des listes parents, et les portions successives sont échangées avec probabilité 1/2. Ce principe est appliqué récursivement, en répétant ce mécanisme dès lors que deux objets de type liste sont présents à la même position dans les segments échangés.

Réparation à base de regexp

Bien sûr, ces opérateurs globaux ne garantissent pas la conservation de l'exécutabilité du code, ce qui implique un *a-priori* de probabilité d'application réduite. Cependant, nous utilisons ces opérateurs dans un contexte évolutionnaire parisien, ce qui implique de toute manière un "conservatisme" important vis-à-vis de la création de structures nouvelles. Et comme chaque requête entraîne une nouvelle génération, obtenir des évaluations fiables implique de changer peu la population (le profil) à chaque requête.

Les opérateurs locaux que nous avons introduits sont très progressifs (muter un terme en un terme synonyme ne modifiera pas profondément le comportement du système en réaction à des requêtes similaires). Il est donc naturel de leur attribuer des probabilités plutôt élevées, du moins respectivement aux valeurs utilisées classiquement dans l'approche parisienne. En comparaison, des valeurs basses de probabilité d'application des opérateurs globaux semblent pouvoir assurer la conservation des caractéristiques importantes des individus tout en permettant une exploration globale raisonnable.

Pour pouvoir tester cette hypothèse dans des conditions réelles, il reste cependant indispensable de disposer d'un mécanisme sûr de conservation ou de restauration de la validité du code. En effet, des probabilités trop élevées pour

les opérateurs globaux, entraînant une proportion d'individus infaisables importante, signifieraient qu'une part importante de la population ne pourrait être évaluée à chaque génération. Nous n'avons alors d'autre choix que de "jeter" les individus concernés, perdant ainsi l'acquis des générations précédente. Il n'est pas possible d'espérer des résultats probants dans de telles conditions, sans parler du surcoût calculatoire.

Un mécanisme de transformation de structure assurant la validité du code résultant doit tenir compte du contexte, soit les prototypes résultants de la section de code précédant et suivant la zone à modifier. Analyser intégralement ces prototypes serait trop complexe. Mais seule une part réduite de ces prototypes nous intéresse. Il s'agit, pour la zone de code en amont, des n premiers types où n est la taille maximale des prototype d'entrée du code à modifier. n est généralement faible si nous cherchons à modifier une instruction, par exemple, et la portion de prototype concernée peut être déduite de l'examen des derniers objets précédant la zone qui nous intéresse. Il en est de même en ce qui concerne la portion de code en aval. Ainsi, un examen local de la structure peut suffire à assurer le succès de la transformation.

Une manière simple de réaliser cet examen, sans analyse proprement dite du code, est de se fixer un "schéma" de code auquel la transformation projetée est applicable. Ce schéma peut être identifié au moyen d'expression régulières, conduisant à une transformation basée sur le principe des réécritures à base de "regexp" bien connues pour la transformation de texte, via des outils comme **Sed** et **Perl**. Pour autant que le schéma d'identification soit réalisé sans erreur (réalisation à la charge de l'implémenteur des opérateurs génétiques...), ce système permet la production de code valide, en assurant le contrôle d'une partie suffisante de l'environnement d'application de la transformation.

Examinons quelques exemples. Un aspect important de la créativité est lié à la présence d' "introns", portions de code totalement ignorées à l'exécution. Un exemple type, dans notre contexte, est le programme

[sous-programme **]** <**DROP**> ,

la dernière instruction éliminant purement et simplement la liste qui vient d'être déposée sur la pile. Une expression régulière identifiant un tel segment, "• <**DROP**>", le "•" pouvant être remplacé par un objet quelconque, permet d'éliminer du code de tels introns.

Les instructions de groupement par opérateurs booléens, permettant de construire des sections de requêtes, sont à l'origine de sections de code équivalentes. Par exemple,

<**SWAP**> <**#NOT**> <**SWAP**> <**#NOT**> <**#AND**>

est équivalent à

<#OR> <#NOT>.

De tels séquences peuvent elles-aussi être identifiés et échangés via des expressions régulières.

Il en est de même des opérations de pile : il est équivalent d'écrire

[scalaire-1 ... scalaire-*k* <**ROLLDN***k*> <*instruction*> <**ROLLUP***k*>]

et

[<*instruction*> scalaire-1 ... scalaire-*k*]

si <*instruction*> ne prend qu'un argument. Une expression régulière permettant l'identification des classes de prototypes des instructions permet d'implémenter cette équivalence.

Comme on le voit dans ces exemples, les tests atomiques dont nous disposons pour écrire ces expressions doivent être de deux sortes : les tests de types des objets, et les tests de prototype des instructions. Puisque le "prototype" d'un scalaire ou d'une liste est vide en entrée et ce type seul en sortie, nous pourrions en réalité ne tester que ce point, mais cela compliquerait l'écriture et la lisibilité des expressions obtenues.

L'implémentation d'un tel mécanisme de remplacement est assez lourd, d'autant plus que nous ne travaillons pas sur des objets "plats" comme des chaînes de caractères, mais que nous devons examiner aussi les sous-listes. Cela se traduit pas un coût algorithmique plus élevé, et surtout une intégration plus difficile dans des systèmes existants. Dans le cas où des probabilités d'application des opérateurs globaux restent basses, une version moins fiable mais plus légère de ce mécanisme semble plus indiquée.

Réparation de programmes : restauration d'arité

Plutôt que de concevoir des opérateurs génétiques laissant la validité du code intacte en toutes situations, on peut envisager une approche plus pragmatique consistant à réparer localement les éventuels dégâts provoqués par l'application des opérateurs. Cette approche, plus légère, donne une complète liberté dans le choix des transformations, laissant en contrepartie certaines constructions invalides sans retouche.

Nous avons développé à la section 2.3 des outils permettant d'identifier un conflit local de prototypes entre un objet et celui qui le suit (conflit qui ne peut avoir lieu que quand ce dernier est une instruction). Nous nous restreignons à la réparation de tels conflits : il s'agit ainsi d'une analyse locale, de coût réduit. Nous utiliserons de plus des techniques simples de restauration de structure :

ajout d'un objet d'un type donné, ou insertion d'une instruction <**DROP**> en cas de type surnuméraire.

Lorsqu'un conflit est détecté, nous identifions le plus petit ensemble d'insertions et de suppressions nécessaire, et insérons les objets appropriés dans le code. Nous avons choisi de résoudre le problème de types d'entrée surnuméraires en utilisant <**DROP**> plutôt qu'une suppression de l'objet lui-même, car cette dernière méthode n'est pas applicable lorsque le premier objet est une instruction, celle-ci pouvant, d'une part, renvoyer plusieurs objets, et d'autre part s'accompagner d'effets de bord.

Ainsi, si nous avons la séquence
["chaîne" <DUP> <MAP>] ,
clairement invalide, car <**MAP**> réclame une liste en premier argument, nous la remplacerons par
["chaîne" <DUP> [ ] <MAP>] ,
programme qui bien sûr ne permet pas à <**MAP**> de réaliser une opération intéressante, mais est au moins valide.

Aucune instruction nouvelle, à part <**DROP**>, n'est ajoutée au code.

Les chaînes introduites sont générées en tirant aléatoirement un terme parmi ceux extraits des dernières requêtes ou des documents visualisés, selon les instruments de mémoire lexicale dont nous disposons.

L'attitude à adopter pour les listes est moins évidente. En effet, générer aléatoirement des listes complexes, ou les créer à partir des modèles dont nous disposons va probablement compliquer inutilement le code, et ne repose en tout cas sur aucun fondement lié à l'algorithme évolutionnaire. Aussi avons nous préféré insérer simplement une liste vide, quitte à la muter en une structure plus complexe à la génération suivante, grâce aux opérateurs génétiques globaux.

Filtrage sémantique à la volée : coût et efficacité

Nous avons examiné, section 3.2, page 91, le rôle important joué par les informations lexicales et sémantiques qu'il est possible d'extraire à chaque génération dans la construction de nouveaux individus en rapport avec les sujets traités et les centres d'intérêt de l'utilisateur. Il est aisé d'obtenir ce type d'informations en récoltant les termes présents dans les requêtes, et en les classant dans un petit thesaurus dynamique. Nous établissons en sus une relation (qui n'est pas ici une synonymie à proprement parler, mais une “communauté de sujet”) entre deux termes extraits de la même requête.

Cependant, le nombre de termes obtenu dépend considérablement des pratiques de recherche des utilisateurs. Si il est important, et suffisant, pour les utilisateurs adeptes des requêtes longues, cette information est trop pauvre pour être réellement efficace dans le cas où l'on ne dispose que de 3 ou 4 mots par requête. De plus, les principes de constructions de requêtes, auxquels sont habitués les utilisateurs de moteurs de TR classiques, tendent à privilégier l'association dans une requête de termes discriminants, donc déconnectés sur le plan sémantique. Aussi la relation que nous établissons a-t-elle peu de valeur.

Une alternative possible est de se baser sur du "vrai" texte, autrement dit les documents eux-mêmes. Analyser les documents visualisés, pour extraire cette information, permet :

- de disposer de relations sémantiques significatives vis-à-vis du contexte documentaire,
- d'obtenir une quantité de termes adéquate (en identifiant les termes clés par extraction lexicale, en fréquence ou en singularité d'usage),
- de renforcer l'information de fitness.

Cependant, cette analyse est coûteuse : l'analyse sémantique de chaque texte représente le coût de plusieurs évaluations de règles. Qui plus est, une telle analyse ne produit des résultats utilisables que sur un document linguistiquement homogène, et est souvent prise en défaut par les publications contenant de nombreux termes de nomenclature.

S'il est possible de dépasser en partie ces limitations en aidant l'analyse par un thesaurus spécifique déjà existant, l'information sémantique extraite est alors fortement conditionnée par celui-ci. En particulier, les termes clés extraits des documents seront toujours des termes présents dans le thesaurus source, ne laissant pas la place à l'innovation lexicale, souvent cruciale dans l'examen de textes scientifiques ou de rapports d'analyse. En outre, cela accroît considérablement le coût de ces analyses, puisque des accès importants au thesaurus sont nécessaires.

3.5 Elise

Nous avons jusqu'à présent examiné les différentes contraintes pesant sur les implémentations possibles du système Elise, et les solutions disponibles. Une part essentielle des performances que l'on peut en attendre réside cependant dans la manière dont sont reliées et associées ces solutions. Le paradigme le plus courant en TR aujourd'hui (requête booléenne donnant lieu à une liste de réponses) doit son succès à sa simplicité, tant du point de vue des technologies employées que de son intégration dans les systèmes de gestion d'information, et de sa prise en

main par l'utilisateur. Nous devons nous inspirer de ce modèle dans la conception de l'architecture d'Elise.

Flux d'information et de données

Un élément essentiel pour faciliter l'intégration d'Elise à des systèmes existants[14] est de pouvoir l'adapter aux technologies et aux contraintes locales. Cela impose d'être en mesure de remplacer les outils réalisant une fonction donnée par ceux préexistant sur le site, lorsqu'ils apportent de meilleurs performances ou qu'ils sont seul à même de garantir l'accès aux données.

Du point de vue de l'architecture, cela se traduit de deux manières. D'une part, le système doit être divisé en "unités de traitement", réalisant une tâche clairement identifiée et non spécifique de notre méthode (par exemple, une tâche sémantique classique, un accès à des données, ou l'examen d'un index). D'autre part, ces unités doivent être reliées par des interfaces limitées à l'essentiel et spécifiées de manière aussi universelle que possible (l'idéal étant l'utilisation de flux dirigés de données, contenant exclusivement les données nécessaires, dans un format répandu).

En pratique, les deux questions ne peuvent être dissociées, mais il est souvent plus rapide d'examiner en priorité les formats et types de données à traiter. L'examen des données indispensables au fonctionnement des composants que nous avons d'ores et déjà identifiés comme indispensables (algorithme évolutionnaire, outil de TR booléen, et interface de suivi des documents visualisés) conduit à dresser une liste de ces données :

- des requêtes
- des populations de modules
- des ensembles de règles de réécriture
- des listes de documents
- des historiques de visite de documents
- des évaluations de règles et de modules
- des listes de termes et leurs relations sémantiques

Présentée ainsi, cette liste ressemble bien évidemment à un inventaire à la Prévert. Celui-ci ne prend son sens qu'en rapport avec une architecture de composants réalisant le traitement et la transformation de ces données. Cependant, avant d'étudier ceux-ci, nous pouvons déjà spécifier les formats impliqués.

Les modules et les règles qu'ils permettent de construire sont des programmes

[14] Notons qu'il s'agit d'un point incontournable pour la réalisation de tests en vraie grandeur, seule manière de mesurer de façon pertinente l'efficacité d'Elise.

OKit. Il ne s'agit pas d'un format standard, puisque développé pour l'occasion, mais l'algorithme évolutionnaire auquel il sert de base reste dans ce cadre un complément à des solutions de TR déjà en place.

Les listes de documents (sans "habillage", ou éléments de présentation) sont aussi des informations directement accessibles dans beaucoup d'outils de TR. Ces listes se présentent comme un flux de vecteurs de chaînes fournissant les éléments caractéristiques de chaque document répondant à la requête. Ceux-ci sont généralement le titre, la date, un identifiant unique, un résumé et une évaluation numérique de pertinence entre 0 et 100. Ces éléments peuvent apparaître dans un ordre variable, et s'accompagner d'indications supplémentaires (date, auteurs, etc...), mais cela change peu de choses au mode de traitement.

Les historiques de visite doivent comporter tous les éléments nécessaires à leur utilisation dans le calculs d'évaluations des modules. Chaque entrée identifie donc un document visité, enregistrant le temps passé sur ce document, et les modules ayant généré une règle de réécriture conduisant à l'inclusion de ce document dans les résultats.

La référence au module n'a de sens que pour une génération du profil, aussi la durée de vie de cet historique est limitée à l'intervalle entre deux requêtes. On peut alors se contenter d'identifier un module par sa position dans le profil, procurant un "numéro d'identification" du module. Cette information doit être passée "de main en main" au cours de la phase de traitement des documents, et il faut donc ajouter un sixième champ aux vecteurs de chaînes identifiant un document : la liste des modules ayant conduit à son inclusion.

Les évaluations de modules sont une valeur numérique associée à chaque module, qui sera prise en compte pour le calcul d'une fitness. Ce calcul pouvant intégrer les évaluations sur plusieurs générations, nous avons choisi d'inclure un historique d'évaluations pour chaque module, directement dans les profils. Le profil se présente ainsi sous forme d'une liste d'entrées, une pour chaque module (leur nombre est donc constant, et correspond à la taille s de population choisie). Chaque ligne comporte les k dernières évaluations et le code OKit du module. Lorsque le module a moins de k générations, les évaluations manquantes sont nulles.

Enfin, en ce qui concerne les informations sémantiques, nous disposons déjà de tous les outils permettant d'examiner et d'interroger des thesauri, cette structure classique en TR est donc naturellement celle que nous adopterons. Il peut cependant être nécessaire de recourir à la forme "floue" que nous avons décrite, en particulier pour des thesauri générés automatiquement.

Les modules de traitement d'information

Maintenant que nous avons identifié les éléments d'information caractéristiques des différents processus à l'oeuvre, nous pouvons construire une architecture, permettant à chaque requête de traiter ces flux et de fournir des évaluations.

Nous connaissons déjà les étapes principales de ce traitement, algorithme évolutionnaire et principe de réécriture, que nous avons détaillé au long de ce chapitre. Ordonner et connecter ces étapes nous conduit à formaliser les autres processus de traitement évoqués, depuis la requête utilisateur jusqu'à la mise à jour du profil.

-1- construction d'une nouvelle population de modules par reproduction et construction à partir de patrons,
-2- production de règles de réécriture par remplacement de modules dans des modèles,
-3- production de requêtes par réécriture de la requête initiale,
-4- extraction de documents pour chaque requête transformée,
-5- fusion et agrégation des listes de résultats,
-6- affichage des listes et suivi des documents consultés,
-7- analyse des historiques pour produire les évaluations.

Afin d'envisager de manière globale les mécanismes impliqués, rappelons brièvement les tâches réalisées par chaque étape.

1- Une nouvelle population est construite à partir du profil courant, en additionnant les individus conservés par élitisme, ceux produits par reproduction (mutation, crossover locaux et globaux), et quelques individus créés à partir d'un modèle, intégrant aléatoirement des termes de la requête.

2- Ces modules produits sont sélectionnés pour être insérés dans des modèles, formant des règles de réécriture complètes.

3- Chacune de ces règles est alors appliquée à la requête de l'utilisateur (exécutée avec cette requête comme argument, puisque cette règle est un programme OKit) pour produire une requête transformée. On conserve la trace des modules ayant servi à cette transformation (nous dirons "responsables" d'une donnée particulière), et cette information sera passée aux étapes suivantes.

4- Chaque requête transformée est passée au moteur de recherche booléen (provenant des outils de TR préexistants applicables à la base documentaire), permettant d'obtenir une liste de documents.

5- Ces listes multiples sont fusionnées en une seule. Les entrées correspondant

à des documents présents dans plusieurs listes sont agrégées, ce qui signifie que les rangs relatifs sont regroupés (par une simple moyenne), et les listes de modules responsables concaténées. Les autres champs, caractéristiques du document, sont bien sûr identiques.

6- La liste unique obtenue est présentée à l'utilisateur, selon le modèle d'interface décrit section 1.2, figure 3.1. Chaque accès à un document de cette liste est minuté, et le triplet (document, modules responsables, temps) archivé.

7- La dernière étape, placée à cette position pour plus de cohérence formelle, est en fait réalisée *après* la requête suivante. Plus précisément, elle est déclenchée par celle-ci, qui indique que les opérations liées à la requête qui nous intéresse sont terminées. Elle consiste à analyser cet historique de consultations, pour produire une évaluation de chaque module. Celle-ci est construite en additionnant les temps passés sur des documents dont ce module est responsable. Elle permettra de calculer les fitness nécessaires à la construction d'une nouvelle génération.

La figure 3.8 présente la structure complète d'ELISE, tenant compte des formats des flux de données évoqués à la section précédente, et de l'architecture modulaire que nous venons de décrire. C'est cette version qui a été utilisée pour les tests présentés au chapitre suivant.

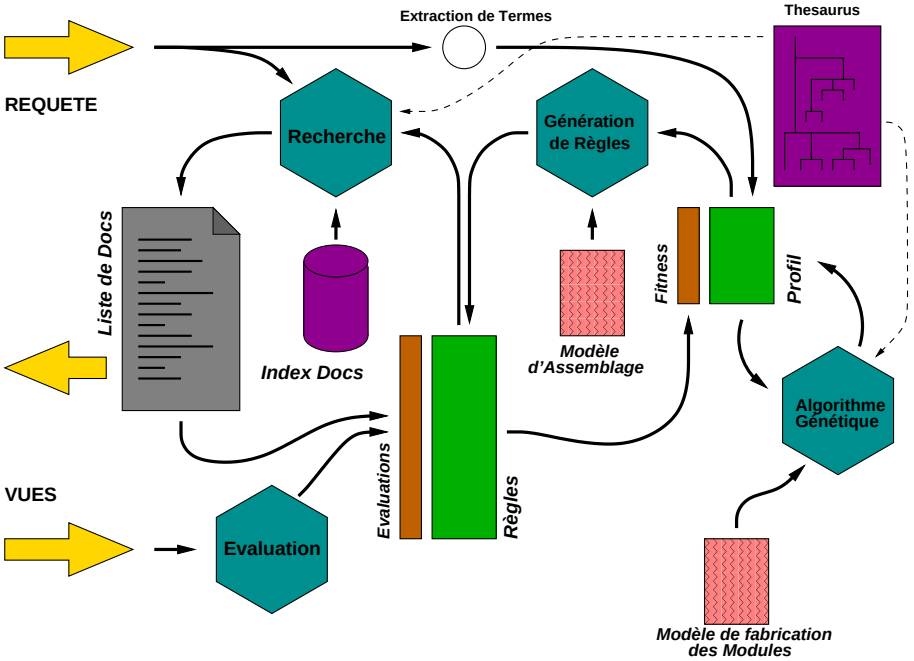
Les coûts, temps caché / temps visible / server-side

Si nous n'avons évoqué que brièvement jusqu'ici les contraintes liées au temps de calcul demandé par la réalisation de certaines étapes, il est nécessaire ici d'en envisager globalement la répartition.

Il est clair que même dans le cas de listes de résultats très volumineuses, l'agrégation de listes de résultats et leur affichage ne nécessitent pas de calculs notables, mais seulement des déplacements de données. Il en est de même de l'analyse des historiques, dont la taille est en outre limitée par le nombre de documents consultés par l'utilisateur (rarement plus de quelque dizaines). La construction de règles de réécriture, si elle nécessite une analyse simple des modèles utilisés, reste elle aussi d'un coût négligeable.

Le temps de construction d'une nouvelle génération est beaucoup plus variable. Il dépend en particulier de manière importante de la complexité des opérateurs génétiques utilisés. L'introduction d'outils de modification de code à base de regexp, au moins dans notre implémentation, pénalise considérablement la performance de cette étape.

Mais dans tous les cas, l'essentiel du temps de calcul est utilisé dans l'exé-



A chaque requête une population de modules est produite, et sert à construire à l'aide de modèles une base de règles, appliquées une par une à la requête originale. Transmises au moteur de recherche, les requêtes réécrites produisent des listes de documents, agrégées et réordonnées avant d'être présentées à l'utilisateur. Les documents visualisés et le temps passés sur chacun d'eux se traduisent en une évaluation de chaque règle, puis de chaque module ayant contribué à cette règle.

FIG. 3.8 – Flux d'information et composants du système ELISE

cution des règles de réécriture, réalisée par la librairie OKit, et le traitement des requêtes booléennes pour obtenir des listes de documents, assuré par un outil de TR indépendant. Si nous n'avons pas la maîtrise de ce dernier, nous avons porté des efforts importants sur l'optimisation d'OKit, au début très pénalisant.

Si le coût global de calcul est un élément important, nous devons garder à l'esprit le contexte client-serveur de l'application. Le temps d'exécution ou d'affichage pour le client, et le volume de données échangé sont deux autres critères importants pour déterminer la viabilité de l'approche. Grâce à l'architecture à base de flux d'Elise, l'essentiel des traitements est réalisé sur le serveur, ainsi les données qui transitent vers les clients sont des données finales, en volumes faibles. Quant aux flux générés par le suivi des consultations, il est naturellement bas. Ces critères permettent l'utilisation d'Elise dans le contexte d'une solution de TR sur un intranet ou sur internet.

Chapitre 4

PROTOTYPE ET EXPERIMENTATION

Résumé du Chapitre.

La validation des modèles et hypothèses examinés plus haut passe par la réalisation d'un prototype fonctionnel. Ce dernier permet en effet non seulement de vérifier leur applicabilité au problème réel que nous nous efforçons de résoudre, mais aussi leur compatibilité avec les techniques et outils existants.

Faute de théorie décrivant de manière quantitative l'influence sur un algorithme de PG des différents paramètres génétiques, le réglage de ceux-ci doit être fait par l'expérimentateur, grâce à des tests sur des benchmarks. C'est *a-fortiori* le cas lorsqu'il ne s'agit pas de paramètres numériques : méthodes d'initialisation, vocabulaires, méthodes de réécriture de code... Il est en outre important d'adapter l'algorithme aux spécificités des bases documentaires utilisées, à l'expertise des utilisateurs, aux sources sémantiques disponibles.

Mais les tests sur des benchmarks ne permettent pas de rendre compte de la réalité fondamentale de l'optimisation interactive. Si l'algorithme s'adapte à l'utilisateur, la personnalité propre et l'adaptation de celle-ci au cours de la manipulation du logiciel ne peuvent être envisagés que lors de tests en vraie grandeur.

Les résultats exhibés par Elise sur les tests classiques utilisés par la communauté du TR (bases TREC, par exemple) comportent des taux de rappel élevés, et des précisions faibles. Si ce résultat est décevant en comparaison des performances des moteurs actuels les plus performants, l'analyse des populations de l'algorithme évolutionnaire montrent un comportement d'apprentissage et de

spécialisation des traitements en fonction du type de requête. La composante sémantique des populations traduit même une capacité d'innovation lexicale, permettant d'accéder à des documents qui ne seraient pas retournés par des outils classiques.

Elise montre qu'il est possible de réaliser un prototype fonctionnel d'un système de personnalisation à base d'EA, en intégrant une solution de TR existante, et des outils sémantiques, pour fournir un comportement de recherche évolutif aux utilisateurs.

1 Paramètres et données

Les algorithmes évolutionnaires s'adaptent aisément à une grande variété de problèmes sans qu'il soit nécessaire de réaliser une étude théorique poussée de ceux-ci. En contrepartie, leur efficacité dépend largement du réglage précis des nombreux paramètres de ces algorithmes (probabilités, tailles d'échantillonnage, modes de calcul de fitness et de sélection). En outre, dans le cas d'Elise, intervient une quantité importante de données liées à la spécificité des structures mises en œuvre (templates, modèles, thesauri).

Toutes les possibilités de réglage et de sources des données ne peuvent pas être testées, mais il est possible d'en éliminer logiquement une grande partie par plusieurs moyens :

- raisons tenant au fonctionnement de l'algorithme,
- analogies avec d'autre cas d'application,
- contraintes techniques rendant la solution trop coûteuse,
- ou simplement absence de sources adéquates.

1.1 Paramétrage

Une part importante des publications consacrées aux algorithmes évolutionnaires concerne la question délicate du paramétrage des "fondamentaux" de l'algorithme : taille des populations, sélection et élitisme, taux d'application des opérateurs génétiques, et calcul de fitness. En l'absence de modèle théorique de ces algorithmes, ces résultats sont essentiellement expérimentaux, qu'il s'agisse d'études exhaustives sur des problèmes de test, ou de cas d'application réels. Nous nous sommes inspirés de nombreux résultats, à commencer par les classiques [Gre86, DeJSpe0]. Cependant, ceux-ci constituent encore une part réduite de la quantité importante travaux publiés dans ce domaine. Aussi ne pouvons-nous prétendre ici ni à l'originalité des positions adoptées, ni à la fidélité à une position déterminée. Par ailleurs, la variété des domaines d'application des AE rend souvent impossible la transposition directe des résultats au domaine de la programmation génétique, et en particulier du text-retrieval par PG.

Tailles de populations et nombre de règles

Les "modules" du profil déterminent la réécriture des requêtes en deux étapes : ils sont d'abord intégrés dans des règles de réécriture complètes, basées sur des templates, avant de participer à la transformation des requêtes lors de l'évalua-

tion de ces règles. Une règle peut intégrer plusieurs modules, et plusieurs règles peuvent être employées.

Il est possible d'envisager un modèle dans lequel une seule règle est produite. Dans la boucle de traitement d'information, décrite section 3.5, cela permettrait d'éliminer l'étape de fusion et d'agrégation des listes de documents obtenues pour chaque requête transformée.

Appliquer cette simplification conduit à un comportement erratique du système : des requêtes successives identiques conduisent à des résultats n'ayant rien en commun, et des modes de réécriture pertinents appris au cours de l'évolution sont oubliés, même dans le cas d'un biais délibéré dans les documents consultés conduisant à un renforcement de ces points. La raison en est simple : seuls quelques modules participent à la construction de la règle unique à chaque génération. Les évaluations sont donc pauvres et concentrées.

Augmenter le nombre de modules participant nous ramènerait à la situation décrite section 3.2, lorsque le profil était constitué des règles elles-mêmes. Ces effets apparaissent, bien que de manière plus modérée, dès que le nombre de règles est petit devant le nombre de modules.

Le coût de transformation des requêtes est directement proportionnel au nombre de règles utilisé, ce qui nous pousse à limiter ce nombre au strict minimum (l'étape d'évaluation des règles compte en effet pour une part importante du temps de traitement de chaque requête utilisateur).

Nous obtenons ainsi un ordre de grandeur empirique pour le nombre de règles produites à partir du profil : celui de la taille du profil, qui est la taille de population de l'algorithme génétique. Des comportements satisfaisants sont obtenus tant que ce nombre reste proche à 10% près de la taille du profil.

La taille de population, qui est le nombre de modules dont nous disposons à chaque génération, conditionne étroitement la variété des requêtes pouvant être traitées. Chaque module est en effet capable de traiter un champ sémantique particulier, et un groupe de structures de requêtes limité. Le premier peut être identifié à l'examen du module, en étudiant les termes qu'il contient. Le second ne peut pas être déterminé aussi simplement, mais est d'autant plus réduit que la structure du module (longueur, nombre de sous-programmes) est simple. Nous savons expérimentalement que les modules les plus complexes ne sont pas utilisables. Leur complexité est souvent le fait d'introns, et la part d'infaisables croit avec la complexité des modules. La plus grande part des modules de fitness élevée obtenus dans les divers types de tests que nous avons conduits reconnaissaient une ou deux structures de requêtes, se contentant d'examiner l'opérateur

booléen racine de la requête.

Aussi, traiter une variété importante de requêtes nécessite l'utilisation de nombreux modules différents. Ce qui pousse à des tailles de population importantes. Cette variété dépend très largement des types d'utilisateurs que nous devons satisfaire. Plus précisément, deux caractéristiques importantes peuvent être mises en relation avec les deux aspects de généralité des modules, sémantique et structurel, que nous venons d'évoquer.

La première concerne le champ sémantique employé par l'utilisateur. Lorsque celui-ci est large, il est possible d'autoriser une augmentation de complexité contrôlée des modules, sans augmentation de taille de population. Pour cela, il faut introduire, parmi les individus initialisés "à neuf" lors de chaque génération, un nombre important de termes sans augmentation du nombre d'instructions ou de sous-programmes. Ainsi, un champ sémantique, ou un nombre de sujets, plus large peut être détecté par chaque module.

La seconde, la complexité structurelle des requêtes, reflète surtout le degré de familiarité de l'utilisateur avec les outils de TR avancés. Si des requêtes complexes permettent dans les outils évolués classiques d'obtenir de meilleurs résultats, c'est souvent le contraire qui se produit avec Elise. Ces requêtes complexes dépassent souvent la capacité d'analyse des modules évolués, et le résultat est alors proche de la simple expansion sémantique, sans les raffinements introduits par la plupart des outils de TR professionnels.

Le seul moyen de contourner ce problème est d'autoriser des modules structurellement plus complexes, en compensant le nombre de modules invalides par une taille de population plus grande. Cette solution est applicable au détriment double (par la taille des modules, et le nombre de modules à gérer) de la performance, et conduit souvent à des délais d'obtention des réponses à une requête au-delà de l'acceptable. Nous avons donc choisi, au moins dans un premier temps, de nous limiter à des requêtes simples, pour conserver des rythmes d'interaction confortables pour l'utilisateur.

Renouvellement, créativité, diversité

Nous avons évoqué à plusieurs reprises la question de la diversité des individus conservés dans les profils utilisateurs, ou, en d'autres termes, l'efficacité de l'échantillonnage de l'espace de recherche. Il s'agit d'un aspect crucial pour le fonctionnement de l'algorithme d'optimisation, pour assurer une exploration suffisante du paysage de fitness, et une réaction rapide aux changements de position des optima. Une diversité importante, permettant un examen complet de

requêtes très diverses, est la condition de la qualité des évaluation des individus, et de la stabilité des indices de performances de l'extraction documentaire. Enfin, contrôler la diversité est essentiel lorsque beaucoup d'individus sont conservés à chaque génération. L'approche parisienne doit ainsi compenser un faible renouvellement des individus par l'introduction pertinente de nouveaux individus.

Conserver et contrôler la diversité impose tout d'abord de définir cette notion, qui repose sur une topologie de l'espace de recherche. Celle-ci est très souvent immédiate lorsqu'on cherche à optimiser des fonctions de données numériques. Dans ce cas, l'espace de recherche est une partie de $\mathbb{R}^n$, et les fonctions à optimiser possèdent une certaine régularité (semi-continues, Höldériennes)[1]. La topologie adéquate pour ces problèmes provient donc de la distance euclidienne, qui fournit aisément une notion de similarité des individus en relation avec la fonction de fitness.

Dans le cas plus "exotique" d'un espace de programmes, définir la similitude de deux individus est beaucoup plus difficile. Intuitivement, deux individus pourront être considérés comme semblables si leur "effet" du point de vue de l'algorithme est proche. Ce qui résulte en deux définitions distinctes selon le niveau auquel on se place pour envisager le fonctionnement du système. Si l'on se focalise sur la tâche de réécriture, deux individus proches transformeront la même requête en des résultats semblables. Si on examine plutôt les ensembles de documents retournés, la proximité se mesure au recouvrement des ensembles documentaires obtenus.

L'espace des requêtes possibles, résultat de la combinaison de l'ensemble des structures d'arbres, et de tous les termes présents dans le contexte lexical, est très grand. Il est donc impossible d'examiner ces deux types de similitude sur toutes les requêtes, et on doit se limiter aux seules requêtes effectivement produites par l'utilisateur. Aussi ces mesures de similitudes sont dépendantes de l'historique des requêtes.

Dans le premier mode de comparaison, basé sur la similitude des résultats de réécriture, la notion de proximité des programmes est transformée en une notion de proximité des résultats. Si cette nouvelle tâche est plus simple, la variété de structures possibles étant considérablement diminuée, elle n'en reste pas moins très vaste, et dépendante du contexte. Examiner la proximité sémantique des résultats en ignorant l'aspect structurel la rend abordable, mais limite l'intérêt de la mesure. Il s'agit donc d'une méthode de comparaison peu satisfaisante.

[1] Notons que l'on ne dispose que très rarement de fonctions continues ou dérivables, car dans ce cas, si l'optimum est unique, les méthodes à base de gradients sont souvent plus efficaces que les algorithmes évolutionnaires.

L'examen des recouvrements entre ensembles documentaires dont sont "responsables"[2] les modules à comparer présente un avantage opérationnel important. Ces ensembles documentaires sont de toute manière déterminés pour la production de la liste finale présentée à l'utilisateur, et une telle mesure n'induit donc pas de calcul supplémentaire. Mais elle n'est disponible qu'après la création d'une nouvelle génération, qui dans le cycle de traitement d'Elise intervient avant tout appel au moteur booléen. Aussi n'est-il pas possible d'utiliser cette mesure lors de l'introduction de nouveaux individus.

Lorsqu'une requête de l'utilisateur est traitée, on obtient finalement pour des modules M1 et M2, des ensembles documentaires D1 et D2. Une définition numérique élémentaire de la distance entre M1 et M2 peut alors être obtenue en calculant :

$$\frac{|D1 \cup D2| - |D1 \cap D2|}{1 + |D1 \cup D2|}$$

Bien sûr, cette formule fournit une distance de deux modules ne fournissant aucun document commun dépourvue de réelle signification. Affiner cette mesure nécessiterait un examen plus fin des relations lexicales entretenues par les documents des deux ensembles D1 et D2, ce qui est trop lourd pour cet usage.

Cette distance n'a de valeur qu'en liaison avec une requête donnée de l'utilisateur. Il est donc inutile de faire appel à des mécanismes de nichage élaborés, alors que la notion topologique que nous utilisons change de manière importante à chaque génération. Nous nous contenterons donc d'utiliser cette distance dans une pénalité appliquée aux évaluations des modules. Considérons une population de modules $M_i, i \in [[1..s]]$. Chaque module M_i est responsable d'un ensemble de documents D_i. L'utilisateur consulte chaque document d pendant un temps $\tau(d)$. L'évaluation pour cette génération de M_i peut être calculée comme :

$$\sum_{d \in D_i} \tau(d) - \alpha \sum_{j \neq i} \frac{|D_i \cup D_j| - |D_i \cap D_j|}{1 + |D_i \cup D_j|}$$

Le paramètre numérique α, qui représente la pénalité par document partagé, doit être pris petit devant le temps moyen passé par l'utilisateur sur chaque document.

Templates et connaissances a-priori

Chaque transformation de requête fait intervenir les règles de réécriture, dont la structure globale provient des "templates" dans lesquelles sont intercalés les

[2] Ce terme est utilisé dans le même sens que section 3.5.

modules du profil. Nous avons vu plus haut qu'un tel mécanisme est nécessaire à la fois pour assurer l'expression efficace de tous les modules, et permettre à ceux-ci de rester d'une taille gérable par l'algorithme. De même, il est nécessaire de recourir à des modèles pour construire les nouveaux individus introduits dans les populations, une génération aléatoire ne permettant pas d'obtenir rapidement des individus efficaces ni d'introduire des éléments du contexte sémantique.

Ecrire ces modèles permet d'introduire les connaissances que nous avons du processus de traitement des requêtes par le moteur de recherche : quelles sont les formes de requêtes acceptables, et comment celles-ci font le lien entre caractéristiques sémantiques et ensembles de documents. Par exemple, nous savons qu'une convergence entre notions s'exprime par la conjonction de sous-requêtes exprimant ces notions via **#AND**. De cette manière, nous fournissons des axes d'exploration de l'espace des transformations de requêtes, à partir de la situation existant à chaque génération. L'approche parisienne, et plus particulièrement les opérateurs que nous avons choisis pour conserver l'exécutabilité du code, donnent lieu à une exploration très progressive de l'espace au niveau individuel. Les deux types de modèles permettent d'accélérer cette exploration sans que cela se fasse au détriment de l'efficacité des modules.

Cependant, il faut se garder d'une trop grande spécificité des opérateurs génétiques ou des modes de génération de nouveaux individus au problème traité. Il s'agit en effet d'une difficulté connue dans le domaine des algorithmes génétiques et plus généralement de l'apprentissage, conduisant à une trop grande spécialisation des résultats obtenus (on parle parfois de sur-apprentissage). Elle conduit à tenter de modéliser le bruit sur les données, en plus du système qui les produit. Cela est particulièrement grave dans le cas où le rapport signal sur bruit est faible, comme cela peut se produire dans le cas d'évaluations provenant d'un utilisateur humain.

Examinons ce phénomène dans un cadre d'optimisation : nous cherchons à optimiser une fonction objectif, pour laquelle nous n'avons accès qu'à une valeur bruitée. En faisant l'hypothèse simplificatrice d'un bruit ergodique, la moyenne de ces évaluations pour un même individu sur un nombre de génération de plus en plus grand converge vers la valeur réelle de la fonction objectif, ce qui permettrait de réaliser une optimisation exacte. Mais si l'on ne conserve qu'un nombre fini d'évaluations, lorsque l'ampleur du bruit est grande, une optimisation "parfaite" conduit à suivre les aberrations dues au bruit, et non l'optimum.

Il est possible de visualiser ce comportement en fournissant à Elise, de manière répétée, la même requête, et les mêmes historiques de consultation[3]. On

[3]il s'agit bien sûr d'un cas de test, une telle situation étant difficilement réalisable dans le

observe alors l'apparition de parties importantes des modules n'ayant, en particulier, aucun rapport sémantique avec les requêtes proposées, et relevant donc probablement de la sur-optimisation, accompagnée d'une diminution des performances (évaluées numériquement grâce au taux de rappel et à la précision, selon la méthode développé section 2). Dans ce cas, utiliser des individus produits aléatoirement plutôt qu'à partir de modèles limite le phénomène, conduisant à un apprentissage plus lent mais sans régression. Ceci conduit à porter une attention particulière à la construction de ces modèles, en prenant garde à ne pas sur-spécifier les comportements.

Les templates permettant de construire les règles à partir des modules doivent comporter au moins tous les cas simples de construction de requêtes par les opérateurs booléens, que nous avons développés section 3.2. Un biais de sélection de ces "templates" (trop nombreuses pour être toutes utilisées à chaque génération) permet de limiter l'occurrence de constructions souvent observées dans les cas de sur-optimisation. Il s'agit en particulier de celles utilisant l'opérateur #NOT, peu utiles pour accéder à un domaine sémantique particulier, mais en revanche très efficaces pour éliminer arbitrairement certains documents.

Les modèles de production de nouveaux individus sont la source d'un biais encore plus important dans la composition des profils. Ces modèles sont indispensables pour produire des individus utilisables en proportion suffisante. Mais la spécificité des tests requis pour l'analyse de requêtes complexes rend impossible la production immédiate de modèles permettant leur réécriture. Il faut alors plusieurs générations pour transformer les individus créés à partir de modèles en outils de transformation efficaces... tâche alors inutile, puisque chaque requête n'est le plus souvent présentée qu'une fois. Réintroduire une proportion faible de modules construits de manière aléatoire (en fixant les probabilités d'apparition des différents types) apporte une solution partielle, à défaut d'être complètement satisfaisante, à ce problème.

Les modèles et templates sont ainsi une zone où le paramétrage de l'administrateur du système est crucial : biais sur les templates, contenu des modèles, proportion de modules aléatoire et composition de ceux-ci, les réglages sont nombreux, et dépendent fortement des comportements moyens des utilisateurs et de la complexité sémantique du contexte.

---

cadre d'une utilisation normale du système.

Principe et réalité du paramétrage

Nous avons examiné, section 4.1 du chapitre 2, les bases et modèles théoriques des AE. Si des résultats de convergence existent, ils ne fournissent pas d'outil simple permettant le paramétrage raisonné des algorithmes. La définition des taux de mutation, crossover, des sélections et modes de renouvellement employés reste de l'ordre de l'expérimentation.

C'est le cas, à plus forte raison, dans le domaine de la PG, où l'espace de recherche est trop complexe pour permettre une modélisation conduisant à des indications de paramétrage. Effectuer les réglage de ces paramètres numériques repose donc sur des tests en conditions réelles, dans lesquels on échantillonne une plage raisonnable de valeurs pour trouver une valeur satisfaisante. En fonction de la complexité de la sensibilité de l'algorithme à ces paramètres, le nombre de points d'échantillonnage requis varie de manière importante. Mais, pour peu que l'on dispose des données et ressources de calcul suffisantes, cette méthode systématique est relativement fiable.

Tel n'est pas le cas lorsque les "paramètres" ne sont pas numériques, comme les modèles et templates dans le cas d'Elise, ou lorsque trop de paramètres numériques sont interdépendants, comme pour les différents ratios de types dans la génération aléatoire d'individus. On en est alors réduit à un paramétrage "à tâtons", ou reposant sur des hypothèses de comportement de l'algorithme relevant de la philosophie du domaine évolutionnaire, voire de l'extrapolation sans fondement.

Si de tels guides sont souvent préférables au tâtonnement total, on doit garder à l'esprit que les algorithmes génétiques sont des méthodes d'optimisation robustes. Il existe généralement une large plage de valeurs ou de formats des paramètres pour laquelle l'algorithme converge, et pour laquelle les temps de convergence restent du même ordre de grandeur. Il est donc possible d'adopter une approche "exhaustive par défaut" du paramétrage, en décidant de donner à l'algorithme lui-même le choix entre toutes les valeurs imaginables. C'est le cas par exemple des approches du type AG auto-adaptatif [Dav89, SchMor87, BenSch00].

Cette approche conduit à des résultats, et permet donc d'évaluer l'efficacité de l'algorithme, en dehors de toute considération de coût algorithmique. Pour améliorer ce dernier, lorsque les autres paramètres ont été réglés de manière satisfaisante, on devra revenir à l'exploration à tâtons des cas possibles.

Dans tous les cas de figure, que l'on dispose ou non d'une méthode systématique, le nombre de tests devant être réalisés est grand. Cela suppose une

quantité de requêtes et d'évaluations de pertinence importante. S'il est possible, pour un système comme Elise, d'obtenir les premières aisément en examinant les traces des utilisateurs dans un système classique, les secondes nécessitent l'implication des utilisateurs, et ne sont disponibles qu'en quantités très limitées.

Il est donc nécessaire de recourir, pour la phase de paramétrage, à des tests automatiques, dont les caractéristiques sont très éloignées des caractéristiques et objectifs de l'utilisation réelle.

1.2 Bases documentaires

L'exploitation pouvant être faite dans le cadre de tests en vraie grandeur (avec des utilisateurs réels) ou sur des benchmarks dépend largement des types d'informations présents dans ces bases, et de leur accessibilité. Nous avons indiqué, par exemple, que disposer d'un résumé de chaque document était important pour permettre un choix avisé des documents consultés. Cela améliore la fiabilité des évaluations récoltées. Si ce résumé est généralement présent dans la plupart des publications scientifiques, une base de documents scientifiques dans laquelle le résumé n'est accessible que dans le corps du documents n'est pas pour autant aisément exploitable.

Ainsi, la facilité d'emploi des bases, et par conséquent la qualité des résultats qu'elles permettent d'obtenir, dépend à la fois de leur contenu informationnel, et de la segmentation de cette information. Nous nous intéresserons donc principalement aux "champs" (dans le sens donné à ce terme dans le cadre des bases de données) disponibles dans les bases documentaires sur lesquelles nous travaillerons.

Segmentation, informations disponibles

Les plates-formes logicielles utilisées pour stocker les documents sont souvent des bases de données ou des structures proches. Ainsi il est possible (et souvent réalisé) d'accompagner le texte du document d'informations concernant son utilisation, sa source ou le contexte dans lequel il a été créé. De nombreux d'éléments d'information "standards" étant par ailleurs présents dans la plus grande partie des documents (du fait de modèles de présentation imposés), ceux-ci sont généralement extraites du corps du document. L'accès à ces informations est alors standardisé pour l'ensemble de la base, et lorsqu'une politique de gestion de l'information numérique est mise en place, une charte documentaire est souvent employée afin d'assurer leur disponibilité. Il s'agit de données comme les auteurs,

la date de création (et, s'il y a lieu, la date d'enregistrement), la division administrative, l'objet (ex : facture, brevet, projet, etc...), une série de mots-clés, etc... Ces éléments, fournis séparément lors de l'enregistrement du document ou extraits de celui-ci *a-posteriori*, sont disponibles individuellement pour chaque document de la base.

Si, dans le contexte d'Elise, ces informations jouent un rôle similaire au corps même du document, et seul est exploité distinctement ce qui tient lieu de résumé, certaines bases proposent aussi des informations sémantiques que nous pouvons exploiter. Celles-ci sont extraites par analyse du document lors de l'indexation, et le coût d'extraction est donc supporté en une seule fois. Nous avons alors accès rapidement à des mots clés, des termes singuliers (utilisés dans un contexte peu fréquent par rapport à l'ensemble des documents), et l'identification d'un contexte sémantique du document (thesauri concernés).

En outre, dans le cadre de benchmarks (par définition réalisés sans intervention humaine), il est nécessaire de disposer d'une liste de requêtes, et d'une évaluation de pertinence de documents. La taille des bases documentaires de test (comme celles servant de base aux TREC [@TREC]) est généralement modeste : quelque centaines de documents, quelques milliers tout au plus. Malgré cela, il est impossible d'indiquer, pour chaque requête et chaque document, un indice de pertinence relative. De plus, ces indices, déterminés à l'avance par un collège d'experts du contexte dans lequel s'inscrivent les documents, sont forcément d'une précision limitée. En pratique, on doit souvent se contenter d'une indication binaire, du type "document pertinent" ou "document non pertinent". Ainsi, les informations de pertinence se réduisent à une liste des documents pertinents pour chaque requête.

Relations croisées et catégorisation

Une autre manière d'exploiter l'information contenue dans les documents est d'analyser les liens qu'entretiennent les documents entre eux, et les catégorisations possibles. Nous venons d'évoquer la catégorisation des documents par contexte sémantique, souvent réalisée pour faciliter l'expansion sémantique et l'analyse, en indiquant les thesauri à utiliser prioritairement. Mais la simple analyse lexicale ne permet pas toujours de réaliser un groupement pertinent des documents en fonction de leur contenu.

L'exploitation des liens explicites (liens hypertexte) ou implicites (références, citations) entre documents fournit une méthode de classification indépendante de la sémantique. Ces liens sont entièrement produits par les auteurs des documents,

et les groupements en découlant n'ont pas la stabilité et la fiabilité résultant d'analyses automatiques comme l'analyse lexicale. Mais ils reflètent plus finement le contexte temporel et cognitif associé aux documents.

Ces classifications sémantiques et relationnelles apportent une contribution importante à la facilité d'exploitation des documents par l'utilisateur, mais ne conditionnent pas de manière impérative les traitements qui leur sont appliqués. En revanche, on ne peut éviter de prendre en compte les catégorisations de langue et de format. Réaliser des traitements sémantiques poussés sur des tables de résultats cliniques est inutile, et utiliser des paramètres d'analyse adaptés aux langues européennes pour des publications en japonais ne produira pas de résultats probants. De plus, certains traitements reposent sur la présence explicite de certaines informations (recherche par auteur, par publication, ...) qui ne sont pas présentes dans tous les documents. En outre, nous utilisons un champ "résumé" pour constituer les listes de résultats de recherche, et permettre une sélection des documents visualisée par l'utilisateur en relation avec leur contenu et non leur place dans la liste.

Dans le cadre des bases de tests, peu de problèmes de ce type apparaissent : les bases sont homogènes, tant dans leur format et contenu, que dans leur registre lexical. L'absence d'un champ résumé n'est pas gênante, puisqu'elles sont par définition destinées à être utilisées de manière non interactive.

En revanche, des tests sur des bases réelles doivent tenir compte de l'hétérogénéité des documents et des thesauri applicables. Dans un cadre industriel, séparer dans des bases distinctes des documents de catégories très différentes (management, recherche, clientèle, par exemple) n'a pas de sens, et rend difficile la communication entre domaines opérationnels. Il en résulte une très grande variabilité de la structure et des contenus sémantiques des documents. S'ajoutent à cela les variations dues à la réunion de bases documentaires résultant de fusions d'activités.

Nous verrons section 1.3 la nécessité d'adapter les thesauri employés aux catégories de documents traitées. Mais de nombreux autres paramètres des outils de TR (bases explorées, tolérance aux variations d'orthographe, sections des documents analysées, langue de travail) possèdent un réglage optimal pour chaque catégorie de documents.

Nous nous sommes limités, dans le cadre du prototype Elise, à remplacer les informations manquantes pour certains documents par d'autres pouvant en tenir lieu (remplacement du champ résumé par un champ de mots clés, par exemple). Et nous avons axé le paramétrage des outils de TR vers une catégorie de documents particulière. L'alternative qui consisterait à laisser ces réglages

à l'utilisateur en fonction de la catégorie de documents qu'il souhaite explorer (comme c'est le cas dans Ulix [VacPar+01]) aurait entraîné une interface de manipulation trop complexe pour permettre une analyse aisée des tests.

Structure des documents

Les bases documentaires que nous utilisons désignent un document par un identifiant unique (numéro d'entrée dans la base, URL, ...) qui permet d'accéder au document lui-même (texte, images, liens...) mais aussi à un nombre parfois important d'informations complémentaires. Nous avons déjà cité, parmi celles-ci, des informations de catégorisation, auteurs, résumé, dates de publication, d'entrée dans la base, etc... plus qu'un document "plat", il s'agit en fait d'une entrée dans une base de données, comportant des "champs" nommés.

Parmi les plus couramment présents, citons **author, abstract, keywords, creation-date, registration-date, references, designation, status**, termes devenus presque standards, tant leur utilisation est généralisée au sein des bases documentaires. Les deux derniers, respectivement la désignation administrative du document (rapport, publication, brevet, offre, etc...) et le degré d'achèvement du document (document de travail, projet, version définitive ou ayant valeur légale...), sont les plus importants pour un outil de TR. Ils permettent en effet la classification du document, et les procédures de traitements applicables.

Le dernier champ présent (souvent baptisé **body**), contenant le document proprement dit, n'est en réalité pratiquement jamais utilisé en dehors de l'indexation et de la consultation par l'utilisateur. La plus grande partie des traitements que nous pouvons réaliser peut se faire sur les deux champs **abstract** et **keywords**. La seule exception notable, une analyse sémantique en profondeur, évoquée section 3.2 du chapitre précédent, demanderait trop de ressources pour être utilisé dans le modèle actuel d'Elise.

1.3 Bases de vocabulaires

Bases spécialisées pour le corpus documentaire

Nous employons dans les opérateurs génétiques et les instructions un thesaurus générique (EmBase, efficace pour les bases biomédicales que nous exploitons). Il permet de ne pas devoir sélectionner d'office, ou faire sélectionner par l'algorithme génétique, les thesauri adaptés à la catégorie de documents recherchée. Cette dernière méthode, de réalisation trop longue pour les tests que nous

effectuons, est cependant la seule solution performante à long terme pour gérer, de manière accessible pour les utilisateurs inexpérimentés, la diversité des bases et des documents. Elle impose de disposer de thesauri spécialisés nombreux.

De tels thesauri sont de toute manière utilisés, en dehors de toute expansion sémantique, par les outils de TR lors de l'indexation comme lors des recherches, pour traiter efficacement les variations de terminologie dues aux domaines techniques concernés. Il s'agit, par exemple, d'assurer la traduction entre formules chimiques, noms scientifiques, et noms commerciaux de substances actives, ou bien entre nomenclatures différentes. Ils sont aussi à même d'assurer les correspondances de vocabulaires techniques entre langues différentes. En effet, les vocabulaires techniques, au contraire du vocabulaire courant, possèdent des traductions précises permettant d'établir ces liaisons.

Nous sommes ici en mesure de réaliser une grande partie des traitements sémantiques à l'aide d'un seul thesaurus général grâce à la spécialisation importante des bases documentaires traités (rappelons-le, il s'agit de bases de publications, analyses économiques ou documents de travail dans le domaine bio-médical). EmBase, ou le MeSH, permettent de cerner efficacement le contexte lexical de ce domaine, grâce à la maturité de ces thesauri, et le travail important consacré à leur maintien à jour[4]. Il reste que leur utilisation efficace, dans un contexte documentaire en évolution rapide, repose sur leur mise à jour régulière. Leur mise en oeuvre au sein de Novartis implique des adaptations "maison" aux bases documentaires, pour parfaire leur adéquation.

Il en est de même des thesauri très spécialisés. Il est nécessaire de disposer de thesauri à jour, parfaitement adaptés aux catégories documentaires auxquels ils sont appliqués, pour que leur utilisation apporte les améliorations souhaitées. Ce qui représente une charge d'entretien élevée.

Informations disponibles, caractéristiques

Les thesauri adoptent des structures assez variables, enrichissant la relation de synonymie formant le coeur de ce type de bases lexicales par des relations parfois très spécifiques du domaine concerné[5]. Les codages numériques, ne répondant pas à des standards précis, varient aussi largement, ce qui fait que

[4] Il s'agit d'une caractéristique particulière du domaine pharmacie et chimie fine, dans lequel le souci des standards et des nomenclatures est très ancien.

[5] Le nombre de relations spécifiées dans les thesauri courants font que l'on devrait plutôt parler de réseaux sémantiques, cependant le terme de thesaurus est généralement utilisé aussi pour ce type de bases lexicales.

l'exploitation de chaque thesaurus nécessite le plus souvent des outils logiciels spécifiques.

Les thesauri extraits automatiquement par analyse des textes forment un cas particulier, car un seul type de relation, que l'on peut baptiser synonymie, peut être extrait. Il s'agit en fait d'une relation de corrélation de l'apparition des termes dans les textes étudiés plus qu'une réelle forme de synonymie (voir Chapitre 2, section 2.2).

Les thesauri spécialisés, extraits de la littérature avec l'intervention d'experts, comportent souvent d'autres types d'information. Dans le cas de thesauri bio-médicaux, comme MeSH, on dispose par exemple de liaisons entre produits pharmaceutiques d'activité ou d'indications similaires, entre pathologies suivant les mêmes mécanismes, etc... Il ne s'agit pas de relations de synonymie réelles, indiquées par ailleurs. Mais ces relations sont utiles pour rechercher des concepts connexes aux termes fournis.

En pratique, on peut se ramener à des structures proches de celles de WordNet [FelMil98] lorsque seule l'exploitation d'un arbre sémantique définissant synonymie, hyponymie et hypéronymie nous intéresse. C'est le cas pour les opérateurs génétiques comme pour les instructions OKit. Nous utilisons donc un codage constitué d'une liste de correspondance terme-concept, d'une table de synonymie entre concepts, et d'un arbre de concepts. Ce type de tables peut être extrait de thesauri comme EmBase ou MeSH, en utilisant des outils appropriés. Leur utilisation est ensuite uniforme, et permet en outre de comparer les résultats obtenus en fonction des thesauri utilisés.

Dans le cas où l'on part de véritables réseaux sémantiques, il serait dommage de se passer de convergences entre termes permettant d'explorer des domaines connexes aux termes de la requête, mais sans relation de synonymie explicite. Il est donc intéressant, pour les besoins de la mécanique évolutionnaire, de "contracter" plusieurs types de relations entre termes en une seule, sorte de synonymie élargie. Ainsi, les notions d'identité de principe actif, d'identité de symptômes ou de population exposée, se traduisent en cette relation lorsque nous utilisons le MeSH. Mais toutes les relations disponibles ne doivent pas être contractées. Il s'agit là encore d'un point de paramétrage devant être validé lors des tests.

Sémantique dynamique, extraction de vocabulaires

On l'a dit Chapitre 1, il existe des méthodes assez performantes d'extraction sémantique à partir du contenu des bases documentaires. Celles-ci fournissent

des thesauri dont le champ lexical est parfaitement adapté à la base traitée, et pouvant être maintenus à jour à un coût raisonnable. Ils ont sur les thesauri issus de travaux d'experts l'avantage d'une très bonne normalisation des relations entre termes, et d'une connectivité élevée. L'exploration par l'algorithme évolutionnaire en est facilitée, lorsque ces thesauri sont utilisés dans les opérateurs génétiques.

Ils ont cependant d'importantes limitations. S'ils sont parfaitement adaptés au contexte documentaire, ils reflètent aussi fidèlement ses biais. Deux termes sans connection sémantique valable, mais très souvent utilisés ensemble pour des raisons techniques risquent d'être mis en correspondance, sans que cela soit compréhensible pour les utilisateurs. Il faut donc garder à l'esprit que ces thesauri présentent une relation de correspondance textuelle, et non de synonymie. Les caractéristiques des thesauri extraits (nombre de termes, et plus encore connectivité) dépendent aussi largement de paramètres numériques. Le calibrage de ces paramètres est difficile, car il n'existe pas de valeurs typiques donnant des résultats corrects sur la plupart des ensembles documentaires. Ils dépendent aussi du langage utilisé, ce qui est un problème dans le cas de bases multilingues. Et ils doivent être contrôlés tout au long de l'évolution du corpus documentaire.

Ces thesauri issus de l'extraction automatique (nous dirons "thesauri dynamiques") sont donc des outils intéressants pour adapter le comportement de systèmes comme Elise aux spécificités documentaires, mais restent des outils complémentaires des thesauri classiques, revus et vérifiés par des experts du domaine.

2 Tests et utilisation : des modèles opposés

2.1 Comment passer d'un modèle à l'autre

Lorsqu'un utilisateur interagit avec Elise, cette interaction se déroule dans les deux sens. L'utilisateur s'adapte à l'outil qu'il utilise, modifie ses requêtes en fonction de résultats qu'il reçoit, autant que Elise adapte ses traitements aux documents consultés parmi ces résultats. Elise s'adapte à un indice de satisfaction qui évolue, et deux recherches identiques peuvent produire des données différentes, tant du point de vue des listes de résultats que des opinions de l'utilisateur.

Lors de tests avec des benchmarks, indiquant quels documents pertinents devraient être retournés en réponse à une requête donnée, il n'y a plus interaction, mais seulement réaction d'Elise à des entrées fixées d'avance. De plus, les listes de documents pertinents sont le résultats de l'examen de la base documentaire

par un collège d'experts. Il ne peut donc apparaître un comportement individuel, des habitudes personnelles, parmi ces données. Et celles-ci sont reproductibles, deux requêtes identiques seront soumises à la même grille d'évaluation, quel que soit l'historique des requêtes les séparant.

Le réglage des paramètres, et même les techniques employées, pour obtenir de bonnes performances dans ces situations différentes varient largement. Quelques points méritent une attention particulière.

Bruitage des évaluations : lorsqu'Elise est testée sur des benchmarks, les évaluations sont sujettes à un bruit beaucoup plus faible que lorsque celles-ci sont produites par un utilisateur. Les solutions pour réduire les conséquences de ce bruit, qu'il s'agisse d'historiques d'évaluations ou de probabilités plus faibles d'applications des opérateurs génétiques, ralentissent l'évolution. Ces paramètres doivent donc être adaptés dans le cadre des benchmarks.

Finesse des évaluations : les bases de tests ne fournissent que des indications binaires sur la pertinence d'un résultat, tandis que nous acquérons lors des consultations de documents un temps de consultation, beaucoup plus fin. Les évaluations dans le cadre des benchmarks sont donc beaucoup plus granulaires, ce qui entraîne des singularités (en particulier liées aux méthodes de sélection, reposant sur un tri) dans le comportement de l'algorithme. On peut notamment obtenir de très grand écarts entre expériences conduites avec les mêmes paramètres.

Hétérogénéité des requêtes : les ensembles de requêtes composant les benchmarks ne relèvent pas d'une "logique" unique : tantôt très détaillées (utilisant de nombreux opérateurs booléens), ou de simples listes de termes composant un sujet vague, elles ne facilitent pas l'émergence de modes de traitement spécialisés des requêtes parmi les modules du profil. Il est souvent nécessaire d'augmenter la taille de population lorsque le style des requêtes est très variable, pour être en mesure de gérer ces différents aspects.

2.2 Résultats des tests : diversité, historique et plafonnement

Evolution des taux de rappel et de la précision : base CFD

La *Cystic Fibrosis Database* (CFD) est une base de test pour le TR, contenant un peu plus d'un millier de documents sur les aspects cliniques, biologiques et pharmacologiques de la mucoviscidose. On dispose en outre d'une centaine de requêtes, et des documents pertinents associés, correspondant à des angles d'analyse variés de cet ensemble documentaire. Cette base, assez ancienne, a

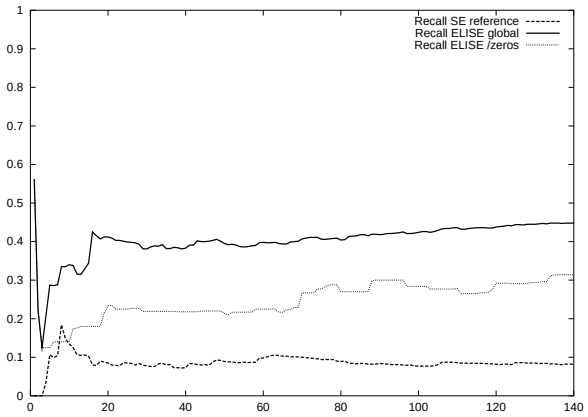


FIG. 4.1 – Taux de rappel moyen cumulé du moteur de recherche booléen (tirets), d'Elise (ligne pleine), et d'Elise quand le moteur de recherche booléen ne donne aucune réponse (points).

été utilisée de nombreuses fois pour évaluer les performances des moteurs de recherche, et nous offre donc de nombreux points de comparaisons. C'est pour cette raison, ainsi que pour la simplicité de traitement des données qu'elle contient, que nous avons choisi cette base.

Pour tester Elise, nous lui avons proposé tout d'abord des requêtes tirées aléatoirement (avec remise) parmi les 100 requêtes disponibles, en suivant l'évolution des taux de rappel et de la précision de deux manières. D'une part, nous avons calculé ces quantités en utilisant les ensembles de résultats retournés par Elise et les ensembles objectif pour la requête courante, à chaque génération. Il s'agit, respectivement, du nombre de documents pertinents retournés, sur le nombre total de documents pertinents, et du nombre de documents pertinents retournés, sur le nombre total de documents retournés. D'autre part, nous avons réalisé la somme, respectivement, des nombres de documents retournés, pertinents, et pertinents effectivement retournés, à chaque génération depuis le début du test. Ces trois sommes ont été utilisées selon la même formule, pour obtenir ce que nous avons nommé taux de rappel et précision *moyens cumulés*[6]. Le but de ce calcul est de rendre les résultats lisibles, en éliminant les fluctuations importantes de génération en génération des taux calculés sur une seule génération, dues à l'aspect stochastique de l'algorithme.

Notons tout d'abord que les performances du moteur de recherche booléen,

[6]Il s'agit de fait d'une moyenne des taux calculés à chaque génération pondérée par le nombre de documents retournés à chaque génération.

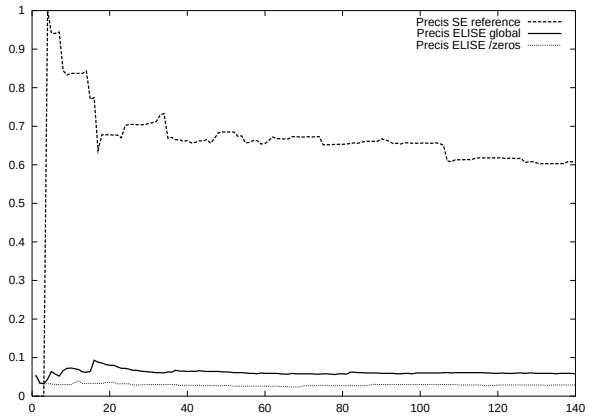


FIG. 4.2 – Précision moyenne cumulée du moteur de recherche booléen (tirets), d'Elise (ligne pleine), et d'Elise quand le moteur de recherche booléen ne donne aucune réponse (points).

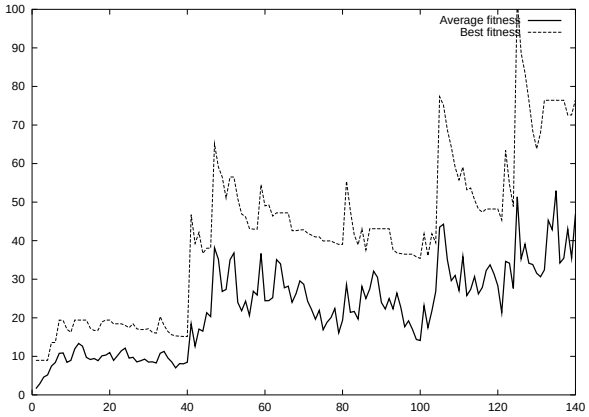


FIG. 4.3 – Evolution de la fitness des modules : meilleur individu (trait pointillé), et fitness moyenne de la population entière (trait plein).

que nous utilisons comme point de comparaison, sont comparables aux résultats obtenus sur cette même base CFD, dans le cadre des TREC, dans la décennie 1990. Nous ne pouvons bien sûr imaginer, à ce stade de la recherche, nous comparer à des moteurs commerciaux, faisant appel à des méthodes fines d'expansion et de recherche.

Les paramètres utilisés tiennent compte des nécessités des multiples contraintes et incitations que nous avons développées plus haut : approche parisienne, fitness bruitée, taille des ensembles de tests réduite, etc... Ainsi, la taille de population utilisée est de 50 (de même que le nombre de règles produites), 70% de la population étant conservé à chaque génération. Le taux de mutation locale est de 30% (par gène, ou atome du programme OKit) et 80% des individus subit un crossover. Les tests sont réalisés sur 140 générations, ce qui signifie qu'une fraction notable des requêtes disponible sera présentée plusieurs fois (le test en comporte 100 en tout).

Les graphes 4.1 et 4.2 font apparaître une nette augmentation du taux de rappel, au détriment de la précision, par rapport aux moteurs booléens classiques. Le taux de rappel atteint 45% dès les 20 premières requêtes, et continue de croître tout au long de l'évolution. Ce comportement d'apprentissage est encore plus net sur la troisième courbe, qui note la capacité d'Elise à contourner les insuffisances du moteur booléen. En effet, non seulement Elise est en mesure de proposer des résultats à une requête pour laquelle le moteur booléen ne propose pas de réponse, mais cette capacité augmente nettement au cours de l'évolution, tendant à rattraper les performances obtenues par Elise dans le cas plus favorable où le moteur booléen propose des résultats.

Elise trouve une plus grande part des documents pertinents, au prix d'une dilution des résultats dans une liste retournée plus large. Cela cadre avec les objectifs que nous nous étions fixés : obtenir les documents difficiles à trouver pour les utilisateurs, qu'il s'agisse de documents possédant une originalité sémantique faisant échouer une simple recherche par termes, ou retourné au sein de très nombreux résultats non pertinents. Il faut cependant noter qu'un tel résultat peut être obtenu en augmentant considérablement les tailles d'ensembles de résultats, ou par une expansion sémantique très large.

Les tailles d'ensembles de résultats restent raisonnables (rarement plus du double du moteur de recherche booléen) dans Elise, mais les mesures de rappel et de précision restent malgré tout peu concluants. L'intérêt de l'approche doit être mesuré selon d'autres critères, et ce d'autant plus que notre intérêt principal était la capacité de personnalisation, autrement dit la spécialisation du système pour répondre efficacement à certains types de requêtes.

L'examen de la composante lexicale (chaînes de caractères) présente dans les modules évolués au bout de 140 générations montre qu'Elise élabore une réelle spécificité au domaine de connaissances exploré. Le point de vue du praticien, destinataire et utilisateur potentiel d'une application d'Elise aux bases biomédicales, s'exprime dans l'analyse du Dr. Thierry Prost :

> Les termes indiqués par le moteur de recherche regroupent notamment un ensemble d'enzymes (ananase, dayto anase, traumanase, bromelains...) utilisées dans diverses pathologies comme anti-inflammatoires. Cet ensemble attire l'attention sur l'utilisation d'enzymes dans la mucoviscidose à la fois comme substitut pancréatique pour compenser la mal-absorption due à l'atteinte de cet organe dans cette maladie et sur le rôle thérapeutique joué dans la fluidification des sécrétions bronchiques (même s'il s'agit dans ce cas d'autres types enzymatiques). L'absence de travaux récents répertoriés dans Medline sur ces enzymes particuliers (les références retrouvées sont antérieures à 1997) est probablement due à l'ancienneté de la base ayant servi aux tests. Si le rapport avec le terme "ciliary arteries" et ses dérivés est moins évident au premier abord (atteinte artérielle fréquente dans le diabète, lui-même fréquent dans la mucoviscidose ?), ce type d'approche a le mérite d'ouvrir au médecin clinicien ou au chercheur des pistes de réflexion ou d'attirer l'attention sur des aspects méconnus de la pathologie étudiée. En raison des masses de données indexées, cette démarche deviendra complémentaire de l'utilisation d'un moteur classique qui vise plus à dresser un état des lieux des connaissances actuelles ou des travaux récents qu'à "tracer de nouvelles voies" de recherche.

Découverte sémantique : la base CFD catégorisée

Si les tests sur la base CFD entière permet de tester Elise dans un cadre générique, il est très difficile d'en extraire des indications sur la pertinence des actions sémantiques (expansion, généralisation, tests de contexte sémantiques) réalisées par les modules. Ils ne permettent pas plus de tester la généralité des opération de transformation structurelles des requêtes élaborées au cours de l'évolution.

Dans ce but, nous avons cherché à diviser la base CFD en sous-ensembles sémantiques, en déterminant des groupes de requêtes s'intéressant à un aspect particulier de la mucoviscidose. La base CFD étant par ailleurs homogène (dans le format des documents, leur taille et leur richesse lexicale), réaliser l'apprentissage

Ensemble n^o	Classification
10	système respiratoire
11	épithélium ou sécrétions du conduit respiratoire
12	pathologies infectieuses et mécanismes immunologiques pulmonaires
13	traitement des affections du système respiratoire
14	autres affections respiratoires
20	conséquences gastro-intestinales et hépatiques de la mucoviscidose
21	syndromes pancréatiques
22	autres pathologies gastro-intestinales et hépatiques
30	conséquences physiologiques et biochimiques dans d'autres organes
40	épidémiologie et génétique de la mucoviscidose
50	diagnostic de la mucoviscidose

FIG. 4.4 – Ensembles et sous ensembles de requêtes de la CFD, classification 1 : pathologies associées à la mucoviscidose, par organe ou fonction.

(l'élaboration d'un profil) sur un sous ensemble et le tester sur un autre permet à la fois de tester la spécificité sémantique et la généralité structurelle des modules produits. Il est ainsi possible de déterminer les facteurs (paramétrage et modèles) permettant d'agir sur ces aspects.

Avec l'aide d'un praticien[7], nous avons réalisé deux classements de ces requêtes selon des axes distincts (voir figures 4.4 et 4.5).

En nous basant sur ces classifications, nous avons réalisé deux types de tests en parallèle. Le premier consiste à fournir pendant 50 générations à Elise des requêtes tirées exclusivement d'un sous-ensemble des questions de la base, constitué de la réunion de quelques-unes des catégories déterminées ci-dessus et couvrant approximativement une de l'ensemble complet des requêtes. Le second consiste à faire évoluer un profil à partir du complémentaire de ce sous-ensemble, pendant 100 générations, puis à spécialiser le profil sur ce sous-ensemble.

A l'issue de ces deux types de tests, nous avons classés[8] les modules obtenus en quatre catégories :

A directement en relation avec le sujet choisi

B ayant un rapport avec le sujet

C globalement en rapport avec la mucoviscidose, mais sans rapport direct avec le sujet

[7] Thierry Prost nous a non seulement aidé pour le classement des requêtes mais aussi pour l'analyse des résultats. Nous ne pourrons assez l'en remercier.

[8] Encore une fois, l'aide de Thierry Prost et ses connaissances médicales nous ont été précieuses.

Ensemble n^o	Classification
10	diagnostic de la mucoviscidose et des pathologies afférentes
20	épidémiologie et génétique de la mucoviscidose
30	aspects cliniques de la mucoviscidose
40	manifestations physiologiques et biochimiques de la mucoviscidose
41	dans le système respiratoire
42	dans d'autres organes
50	traitements de la mucoviscidose
51	traitement des manifestations pulmonaires
52	traitement des conséquences digestives
53	autres traitements
60	autres pathologies et manifestations

FIG. 4.5 – Ensembles et sous ensembles de requêtes de la CFD, classification 2 : spécialités de la pratique médicale.

D sans rapport avec la mucoviscidose

Les résultats obtenus sont présentés table 4.6.

Il apparaît une corrélation entre taille de l'ensemble d'apprentissage et taux de rappel, alors que lorsque ceux-ci sont élevés, on note une explosion de la proportion de termes connexes, mais non directement reliés, au sujet de test choisi. Cela implique une relative capacité du système à apprendre les caractéristiques sémantiques de la CFD dans son ensemble, et non seulement de l'ensemble d'apprentissage. Le modèle partiel élaboré sur un ensemble d'apprentissage de taille suffisante permet alors de réaliser un traitement approprié d'un domaine connexe, mais non étudié.

L'analyse du Dr. Thierry Prost souligne cette capacité à l'extraction de notions connexes au sujet traité :

> Certains mots-clés renvoyés par cette deuxième série de tests mettent l'accent sur des aspects non classiques ou indirects du traitement de la mucoviscidose qui n'auraient pas forcément été abordés dans une approche habituelle. En effet, on retrouve des références pertinentes portant sur des sujets aussi divers que le cathétérisme, l'éducation, l'exposition au tabac ou les organismes génétiquement modifiés ; il est à noter que ces items "inhabituels" sont majoritaires par rapport à des références à la fois pertinentes et "incontournables" telles que par exemple les infections. (Pseudomonas aeruginosa, adénoviroses humaines, sont des termes renvoyés par le moteur)
>
> D'autre part, si l'on classe les items selon qu'ils intéressent plus spé-

Numéro de test	Run 35	Run 36	Run 37	Run 38[2]	Run 39[2]
Ensemble d'apprentissage[1]	1-13	1-12	1-!12	2-50	2-!50
Taille de l'ensemble d'apprentissage	14	14	85	21	81
Ensemble de test[1]	1-13	1-12	1-12	2-50	2-50
Taille de l'ensemble de test	14	13	13	21	21
Taux de rappel	38.1	38.4	42.3	33.3	35.4
Précision	9.8	10.4	12.2	12.0	9.1
% de termes A	19.2	28.6	26.6	20.0	28.6
% de termes B	19.2	57.1	40.0	10.0	42.8
% de termes C	11.6	0	20.0	30.0	5.7
% de termes D	50.0	14.3	13.3	40.0	22.8

FIG. 4.6 – Performance d'Elise sur les bancs de test, et classification de la pertinence de la composante lexicale des modules en fin de test.

Note 1 : La notation "N-S" signifie "Classification N, ensemble S". "N-!S" signifie "Classification N, toute la CFD *sauf* l'ensemble S".

Note 2 : Les classifications de pertinence pour les tests 38 et 39 sont dus au Pr. Gilles Roger

cifiquement les cliniciens ou les chercheurs en biologie fondamentale, on trouve environ un tiers de termes du domaine de ces derniers classés dans la catégorie "undirectly related to the test set" et la moitié classés dans la catégorie "unrelated terms", il sera intéressant de tester le moteur en conditions non simulées afin de voir s'il sépare clairement ces deux types d'utilisateurs et réduit ainsi la proportion de termes non pertinents.

2.3 L'influence des tailles sur les indices et sur la performance

Nous avons examiné les implications possibles, en termes de coût algorithmique, ou en terme de fonctionnement de l'algorithme évolutionnaire, des tailles caractéristiques des populations et des bases de règles. Les règles dégagées fixent des bornes utiles à ces valeurs, de manière assez large. Pourtant, sur le plan pratique, des variations de l'ordre du dixième de ces valeurs influencent déjà de manière sensible le comportement de l'algorithme.

Ce type de réglage fin ne peut être fait que "à la main" : en testant plusieurs combinaisons de valeurs, et en examinant les résultats obtenus. Mais il faut se garder de trop de zèle. Changer de base documentaire, ou simplement de base de tests (l'ensemble des requêtes posées) comme nous l'avons fait pour obtenir les

résultats présentés plus haut, suffit à rendre de tels réglages obsolètes. Autrement dit, ces réglages ne peuvent être faits finement pour plusieurs utilisateurs.

Nous avons adopté, dès le début, une approche basée sur des tailles de population fixe, pour la simplicité d'analyse et de développement qu'elle apporte. Il apparaît que la seule manière de permettre un paramétrage efficace de ces quantités, pour un groupe d'utilisateurs, est de le faire réaliser par l'algorithme lui-même. Si nombre de solutions de ce type ont été déjà utilisée avec succès dans d'autres contextes, leur transposition au contexte de la PG parisienne n'est pas immédiate.

Il reste que ces tailles influencent de manière drastique les coûts algorithmiques de traitement d'une requête, directement transposable en coût de maintien d'une version industrielle éventuelle d'un tel logiciel. L'adaptabilité et le champ fourni aux ajustements de l'administrateur ou à une méthode de paramétrage automatique doivent donc être limités à un intervalle raisonnable. Dans le cadre expérimental dont nous disposions pour les benchmarks, comme dans le cadre des essais préliminaires sur une base biomédicale volumineuse, il était difficile de dépasser des tailles de populations de 80.

Il reste possible d'examiner l'effet de tailles supérieures dans les premières générations. Une taille de l'ordre de 100 permet d'améliorer la diversité (mesurée en termes de diversité lexicale). Au-delà de 120, plus aucun progrès n'est constaté. Des tailles de populations inférieures à 30 entraînent une très nette dégradation des performances, voir leur stagnation complète.

Ces tests partiels nous permettent de proposer un intervalle de tailles de population se situant entre 30 et 80, dans le cadre de fonctionnement que nous avons étudié. Un examen de l'influence du nombre de règles produites à chaque génération, dont nous avons déjà vu qu'il est très corrélé à celle des populations, conduit à des valeurs entre 30 et 50. La valeur supérieure plus basse s'explique par l'impact plus important sur les coûts d'exécution de cette base de règles.

3 Analyse des tests

3.1 Comparatif de résultats, benchmarks habituels et objectifs

La base utilisée pour les tests présentés plus haut, la CFD, date du début des années 1990. Elle a été utilisée pour tester des moteurs de recherche booléens, comme celui que nous employons pour interroger les index documentaires. Sans intervention d'une quelconque expansion sémantique, ni d'heuristiques de recali-

brage des index, les performances de ces moteurs sont très similaires à celles du moteur booléen que nous utilisons.

L'évaluation classique des moteurs de recherche repose sur l'évaluation de la précision pouvant être obtenue, à taux de rappel fixé (typiquement, autour de 10%). Les valeurs obtenues sont généralement supériques à 80%. Aussi, tandis qu'Elise dépasse largement les impératifs de taux de rappel des tests classiques, ses résultats en précision semblent-ils très décevants. Le prototype actuel est effectivement incapable d'obtenir, dès l'initialisation d'un profil, donc sans retour d'information de l'utilisateur, des résultats en ligne avec ceux obtenus par les outils classiques. Mais les outils classiques, quant à eux, ne parviennent pas à approcher les taux de rappel d'Elise, nécessaires pour une exploration en profondeur de la base documentaire. Elise est en mesure de s'acquitter de la tâche mettant souvent en échec les moteurs classiques : retrouver un document précis, ou un aspect précis d'un sujet, dans un domaine plus vaste très représenté dans la base.

3.2 Positionnement : une preuve de faisabilité

Elise montre des capacités d'adaptation à un contexte sémantique et cognitif, et est capable de généraliser les propriétés de ce contexte. Ce sont des composantes essentielles du moteur de recherche personnalisable et évolutif que nous cherchions à créer. Ce prototype montre qu'il est possible de réaliser un tel système, de manière compatible avec les outils existants, et que l'on peut réutiliser une part importante de l'infrastructure de gestion de la connaissance déjà en place. Il valide aussi la faisabilité d'une approche à base d'algorithmes évolutionnaires, et souligne les avantages de l'évolution artificielle interactive pour permettre l'adaptation des logiciels à leurs utilisateurs.

Pour amener Elise à des performances comparables avec les systèmes de recherche les plus avancés, de nombreuses voies d'amélioration s'ouvrent à nous. Sur le plan technique, l'utilisation d'outils plus performants amènerait une plus grande stabilité et fiabilité des résultats. Sur le plan stratégique, une meilleure intégration de l'outil évolutionnaire avec la sémantique et l'extraction d'information permettrait une analyse plus profonde tant des documents que des requêtes.

Chapitre 5

CONCLUSIONS PERSPECTIVES DE RECHERCHE

Le domaine de la personnalisation des moteurs de recherches, et de la réponse aux particularités individuelles a été peu exploré par la communauté du TR, tant sur le plan pratique que sur le plan théorique. L'approche qui consiste à améliorer jusque dans les détails les structures monolithiques existantes semble encore plus rentable. Celle-ci semble cependant se heurter à des limites inévitables : des volumes de données trop importants, et une plus grande attente des utilisateurs, pour lesquels les systèmes de TR sont devenus des outils de travail irremplacables.

Si les performances obtenues par Elise ne peuvent pas encore se comparer aux performances de ces systèmes très évolués, ils montrent pourtant des capacités novatrices : adaptivité, réactivité, et innovation dans les documents retournés. Le système actuel reste un protoype, et beaucoup d'aspects importants n'ont pas ou peu été explorés, faute de temps.

1 Prise en compte du contexte documentaire et ergonomie

1.1 Améliorer la structuration des informations dans les bases documentaires

Il est difficile d'utiliser des informations provenant de la base documentaire pour diriger l'évolution dans Elise. Un des principaux freins en est la structuration insuffisante de ces informations. Il est alors difficile de catégoriser efficacement ces informations sur le plan sémantique, et de mettre en relation des informations de même type pour des documents différents.

La "fitness interne", caractéristique essentielle de l'approche parisienne, est actuellement déterminée par le biais d'heuristiques simples, calculées sur un seul document. Une comparaison des documents sélectionnés par l'utilisateur entre eux, et la corrélation des information qu'ils contiennent, serait plus fiable et refléterait mieux la situation de la population de modules dans son ensemble. Plus encore, le choix d'un document par l'utilisateur relève bien souvent d'une telle comparaison, et cette mesure est donc proche des comportements que nous tentons de modéliser.

Des indications de pertinence des mots clés extraits, ou bien encore l'indication qu'un document appartient à un ou plusieurs groupes de similarité sémantique, permettrait de relativiser l'évaluation fournie par l'utilisateur, et de détecter les modules à l'origine d'une catégorisation sémantiquement pertinente. Il serait alors possible de déterminer de manière fiable une fitness interne, et donc d'utiliser plus efficacement les précieuses informations de consultation des documents.

Décider d'exploiter un "champ" particulier des informations liées à chaque document se heurte souvent aussi à l'absence d'uniformité de ces données, voire à l'absence de ces informations pour certaines catégories de documents. Si les champs extraits du corps des documents sont quant à eux disponibles et uniformes, l'information qu'ils portent est souvent peu pertinente, à l'exception des champs fournis par extraction sémantique.

Elise bénéficierait d'une utilisation plus précise des capacités de segmentation et de mise en relation des systèmes de TR modernes :

- emploi de groupements de documents par "grappes" sémantiques, basés sur des rapprochements lexicaux, pour calculer une fitness interne de meilleure qualité,
- création de nouveaux individus à partir des profils lexicaux des documents

visités.

1.2 Exploiter au mieux les outils sémantiques

L'analyse sémantique effectuée lors de l'indexation par la plupart des solutions de TR est très limitée, en raison de temps d'analyse très élevés. Il s'agit le plus souvent d'extraire des vocabulaires spécifiques de la base documentaire, ou de produire un champ de mots-clés pour chaque document. Elise peut utiliser ces informations pour un meilleur calcul de fitness, pour orienter l'évolution, ou pour réaliser un paramétrage automatique (voir Chapitre 3, section 1.3). Dans tous les cas, il s'agit d'informations comparatives, entre documents visités ou entre ceux-ci et les modules du profil. Des informations pré-calculées, indépendantes du contexte de recherche, telles celles pouvant être accessibles dans les champs d'une entrée documentaire, ne sont donc pas suffisantes.

Les outils d'extraction et d'analyse lexicale restent cependant trop gourmands en temps de calcul pour permettre une utilisation en temps réel sur les documents consultés à chaque requête[1]. A fortiori des méthodes d'analyse plus poussées comme LSI. L'intégration dans Elise d'outils d'analyse à la fois efficaces et rapides, même au prix de l'utilisation de thesauri adaptés, reste donc un problème ouvert.

Les outils actuels permettent une plus grande flexibilité de l'utilisation des outils sémantiques, aussi grâce à la possibilité offerte à l'utilisateur de gérer lui-même quels outils sont utilisés. En particulier, sélectionner les outils sémantiques utilisés pour l'expansion permet d'orienter la recherche et les analyses vers un sens ou un aspect particulier d'un problème. S'il est difficile d'offrir cette possibilité dans Elise, en sus de l'optimisation des profils, confier cette tâche, requête par requête, aux règles de réécriture elles-mêmes peut être un bon moyen d'étendre leur champ d'action. Cela peut se faire grâce à des instructions spécifiques, déclenchant la sélection d'un thesaurus particulier, explicitement ou de manière dépendante du contexte.

1.3 Ergonomie et récolte d'information

Malgré les remèdes que nous avons apportés, la fiabilité des évaluations récoltées et leur nombre restent faibles, diminuant la performance de l'évolution (les informations permettant de la diriger étant partiellement perdues dans le bruit). Nous avons aussi évoqué de nombreuses améliorations pouvant être mises

[1] Voir les contraintes matérielles et de performance évoquées au chapitre 2, section 3.2.

en place à condition de disposer d'outils d'analyse sémantique performants et rapides.

Mais la qualité de ces évaluations est conditionnée au premier chef par la simplicité de manipulation du système, et une bonne information de l'utilisateur à toutes les étapes de cette manipulation. Etudier l'ergonomie des interfaces d'Elise semble donc une étape importante dans la réalisation d'une interaction efficace entre système d'apprentissage et utilisateur.

La présentation des listes de résultats est dirigée par la grande habitude qu'on les utilisateurs des formulaires de recherche présents sur le web (Google, Alta-Vista, etc.. ont des interfaces similaires, et celles-ci sont devenues un standard de fait). Proposer des informations supplémentaires à la demande (un résumé plus complet, des statistiques d'utilisation de termes), comme celles disponibles à présent dans Ulix [VacPar+01] peut constituer un moyen très efficace d'enrichir l'information des utilisateurs et de récolter des indications partielles d'intérêt pour un document.

2 Outils de réécriture et d'analyse des requêtes

Une composante importante de la performance algorithmique d'Elise est contenue dans la structure du langage de réécriture des requêtes, OKit. L'ensemble d'instructions associé détermine quant à lui les opérations réalisables par les modules et leur complexité d'expression.

2.1 Etendre OKit pour traiter des arbres, suivi de typage

Dans le prototype actuel, les requêtes des utilisateurs et les résultats de leur réécriture ont codés sous formes de listes imbriquées (similaire au codages des arbres en Lisp). Il s'agit de la même structure que celle utilisée pour construire des sous-programmes. L'absence de distinction entre programmes et données (il s'agit de listes dans les deux cas) permet une plus grande complexité des formes reconnues et traitées par les modules. Cet effet contribue à répondre à notre attente initiale d'un langage permettant de traiter des données plus complexes.

Cependant, nous avons vu que cette identité entre programmes et données pouvait être à l'origine d'une grande part des modules invalides. Si une proportion d'invalides peut être désirable, selon les conditions, il est souhaitable de disposer du plus grand contrôle possible sur cette quantité. Disposer d'un type distinct "*arbre*" dans OKit peut fournir ce contrôle.

Nous avons décrit plusieurs mécanismes de contrôle de validité du code des modules et des règles (analysés plus en détail en annexe). Ceux-ci se basent sur des informations d'arité des instructions, et réalisent un test qui ne demande pas de disposer des arguments explicites du programmes, et reste moins coûteux que l'exécution réelle. Mais l'analyse de programmes comportant des sous-programmes reste impossible. Une meilleure définition des informations attachées à chaque instruction, et des outils dépassant l'obstacle des sous-programmes, permettraient aussi un meilleur contrôle des modules invalides.

Plus important, ils pourraient servir de base à des opérateurs génétiques plus sensibles au contexte syntaxique dans lequel ils opèrent. Cela ouvrirait la voie à une extraction et une conservation explicite de sous-programmes réalisant des opérations de modification de requêtes ou des traitements sémantiques particulièrement performants.

2.2 Règles de réécriture et efficacité des instructions OKit

Un obstacle important à la réalisation de tests à grande échelle d'Elise, avec des utilisateurs réels, est le temps encore long de traitement d'une requête, malgré l'attention que nous avons portée à l'optimisation de ce traitement. Un délai pouvant aller jusqu'à une minute pour obtenir la page de résultat rend l'utilisation pratique d'Elise encore rébarbative. Deux points particuliers sont responsables de ces temps de réponse : la réécriture des requêtes, et l'interrogation des index documentaires. Le système de TR utilisé doit en effet traiter une requête par règle de réécriture produite, ou une très grosse requête par génération, selon la solution retenue.

Le coût de la deuxième étape dépend du nombre de règles produites. On a vu que celui-ci devait rester du même ordre de grandeur que la taille de population, elle-même fortement imposée par des impératifs de créativité et de conservation de modules efficaces. Nous évoquerons plus bas un axe de recherche permettant d'agir sur ces deux plans en conservant certains modules "intéressants" pour l'évolution.

La première étape, la réécriture des requêtes, est réalisée par la bibliothèque OKit, et les instructions de base du langage. L'exécution se fait actuellement à l'aide d'une machine virtuelle, qui interprète un code binaire issu de la compilation des programmes "lisibles" (source OKit). Compiler directement dans l'assembleur natif de la plate-forme d'exécution pourrait permettre de gagner un facteur 200 à 500 dans les vitesses d'exécution, ce qui serait probablement un grand pas vers un prototype manipulable par toutes les catégories d'utilisateur.

3 Algorithmes évolutionnaires et interaction

3.1 Bibliothèques de modules immortelles : comment discriminer ?

Les calculs de fitness des individus (modules) dans Elise intègrent un historique des évaluation passées, pour stabiliser le contenu du profil, et assurer la conservation des modules exécutant des transformations efficaces. Mais après le traitement d'une série de requêtes portant sur un sujet qui ne permet pas l'expression de ces règles, ces modules sont souvent éliminés de la population.

Un moyen de remédier à cette perte des acquis de l'évolution est de créer une bibliothèque de modules, où seront conservés les modules ayant atteint des fitness élevées, ou des portions de ceux-ci. Intégrer une telle "mémoire" à Elise impose de répondre à deux questions : comment effectuer le choix des modules à conserver, et combien de temps les conserver.

La manière la plus simple de décider des modules à conserver porte sur la fitness obtenue à chaque génération. Garder les modules dépassant un niveau de fitness préétabli se heurte à la difficulté de déterminer à priori un tel seuil. En effet, les fitness ne sont pas normalisées, et la valeur maximale qu'elles atteignent dépend du nombre maximal de résultats dans les listes de documents retournées. Un autre mode de décision basé sur les fitness consiste à garder les modules dont la fitness dépasse d'une certaine proportion le niveau de fitness moyen. Si l'on n'a plus ici de problème de normalisation, il reste que les irrégularités temporelles des fitness sont trop importantes pour que ce choix soit fiable. Des modules de mauvaise qualité sont susceptibles d'atteindre, sur une requête particulière (mal formée, vague par exemple) des fitness élevées, faussant ainsi la sélection.

On peut aussi effectuer ces choix en se basant sur l'originalité des formes obtenues. Un niveau minimal de fitness reste indispensable pour valider l'éligibilité du module, mais la décision finale repose sur l'originalité des traitements qu'il effectue. Il s'agit alors, avant tout, de favoriser l'exploration. Mais cette méthode repose sur une mesure de distance des modules, dont on a vu qu'elle était difficile à établir, et source de biais difficilement contrôlables.

Il faut aussi disposer d'un critère d'élimination de certains des modules conservés, pour assurer le renouvellement de la bibliothèque, sans que sa taille explose. Le critère le plus intuitif consiste à éliminer les modules ou parties de modules qui ne conduisent pas à des fitness élevée lorsqu'ils sont réintroduits dans la population. Le risque est alors une trop grande similitude de ce critère avec celui conduisant à la conservation ou non d'un individu dans la population.

Ce qui revient à augmenter la longévité des individus, possibilité déjà testée sans résultat positif. La méthode "aveugle" d'élimination aléatoire pourrait s'avérer en fin de compte la meilleure.

3.2 Interaction : server-side ou client-side ?

Le modèle actuel d'Elise effectue tous les traitements d'information du coté du serveur de recherche. Les profils eux-mêmes, éléments définissant la "personnalité" de la réponse apportée par le système aux requêtes de l'utilisateur, sont stockés sur le serveur.

La conception modulaire, basée sur des flux de données, permettrait de transférer les profils, et une partie du traitement des requêtes (production de règles, réécriture, et évaluation) du coté du serveur. Il est même envisageable de faire appel à plusieurs outils de recherche booléens en parallèle, et donc d'interroger plusieurs bases documentaires. On pourrait alors utiliser Elise comme une "sur-couche" sur des moteurs de recherche classique.

Cependant, tous les moteurs ne sont pas compatibles, et utiliser le même profil pour tous impose une standardisation des informations retournées. Tout d'abord, les opérateurs booléens disponibles n'incluent pas toujours des extensions comme l'opérateur de proximité. Certains moteurs réalisent une transformation des termes de la requête (expansion, lemmatisation, élimination de "stopwords"), d'autres pas. Les informations accessibles incluent parfois un résumé, parfois une simple liste de mots clés. Enfin, les information des "rang" (une évaluation empirique et automatique de la pertinence du document) sont obtenues par des modes de calcul extrêmement variables. Il est donc nécessaire de réaliser un réordonnancement des résultats adéquat si l'on ne veut pas biaiser l'évolution.

4 Outils d'analyse, de paramétrage et d'administration

Le réglage des paramètres dans Elise, réalisé actuellement à la main, doit pouvoir être fait de manière semi-automatique pour permettre une utilisation industrielle d'un tel système. De plus, les interventions des administrateurs dans ce paramétrage, et dans la liaison avec les bases de vocabulaires, doit être dirigée par des indications pertinentes issues du système en cours d'utilisation.

4.1 Suivi de l'évolution et statistiques

Les profils utilisateurs sont au coeur du fonctionnement d'Elise. Toute information destinée à permettre le suivi du système doit donc inclure une information globale sur les profils. S'il est très difficile de dégager des tendances quant aux techniques de réécriture ou aux sous-programmes, l'aspect sémantique est d'abord beaucoup plus aisé.

Nous disposons d'outils performants pour réaliser cette analyse, permettant d'obtenir des vues statistiques détaillées. Leur coût algorithmique n'étant pas ici un problème, on peut envisager de dégager des tendances d'utilisation de termes ou de domaines lexicaux, en mettant en évidence des sous-groupes d'utilisateurs. Cela permettrait par exemple de paramétrer de manière distincte l'algorithme évolutionnaire en fonction des modes d'utilisation des groupes mis en évidence.

4.2 Automatisation du paramétrage

Nous avons déjà brièvement évoqué l'automatisation du paramétrage (en particulier, la détermination des probabilités des opérateurs et des tailles de population) dans le cadre des algorithmes génétiques classiques, à base numérique. Réaliser celle-ci pour Elise rajouterait un niveau de complexité au système, mais est certainement nécessaire dans l'optique d'une utilisation industrielle, dans des contextes variés. A fortiori dans l'hypothèse évoquée plus haut d'une sur-couche générique aux moteurs de recherches, installée chez chaque utilisateur.

Une approche explorée avec succès dans d'autres cadres est celle des AG auto-adaptatifs, dans lesquels les paramètres sont codés dans le génome, et font l'objet de modifications à chaque génération. Si cette technique est d'application délicate (de nombreux tests sont souvent nécessaires avant de parvenir à des méthodes légères et efficaces), elle permet de limiter au maximum l'intervention humaine dans le paramétrage du système de recherche.

Dans le prototype actuel, beaucoup de paramètres globaux (en particulier les modèles de création de nouveaux individus et de règles de réécriture) sont partagés entre tous les utilisateurs. Nous avons souligné la difficulté et l'arbitraire du paramétrage de ces éléments. Si l'on cherche à produire automatiquement ces données, deux voies s'offrent à nous : améliorer ces paramètres utilisateur par utilisateur, ou au niveau global. La première conduit à une personnalisation totale d'Elise, la seconde implique de faire converger les données et les résultats obtenus pours les différents utilisateurs, débouchant sur des notions d'optimisation collaborative.

5 Aspects collaboratifs et perspectives en intelligence distribuée

L'analyse de profils obtenus sur des tests en grandeur nature (par exemple au sein de la plateforme Ulix) montrera probablement des similitudes importantes entre les profils obtenus par des utilisateurs explorant les mêmes domaines documentaires. Il semble intéressant, dans ce contexte, de regrouper les résultats obtenus pour tirer parti de la plus grande densité d'évaluations et des nombres d'individus plus importants permis par un groupe d'utilisateurs important.

Plus généralement, cela lève la question des synergies qu'il est possible de faire apparaitre au sein de sous-groupes d'utilisateurs, voire au niveau du système global. Les modules sont dans Elise des "briques" de construction de règles dont on espère qu'ils remplissent des fonctions utiles, générales et réutilisables. Il semble très probable que les modules produits dans le profil d'un utilisateur particulier seront – en partie – utiles dans les profils d'autres utilisateurs. Ce qui amène à l'idée de constituer des bibliothèques de modules évolués, non au niveau d'un seul utilisateur, mais à l'échelle d'un groupe ou de la totalité des utilisateurs, afin de faire profiter à tous les résultats de chacun.

Cette proposition rejoint la notion de filtrage collaboartif. Basé sur la détection de consensus d'opinions (le plus souvent explicites), il pourrait trouver une généralisation dans la détection de modules partagés au sein d'une communauté de profils. Il s'agirait alors d'élaborer en commun des méthodes de traitement de documents, s'appliquant maintenant à l'ensemble des ressources, fournissant une réponse efficace au filtrage dans un environnement très variable.

Nous avons cité plus haut et développé quelques-unes des voies de recherche pour l'amélioration d'Elise. Mais la complexité du système permet de trouver des possibilités d'amélioration à de nombreux niveaux, qui relèvent le plus souvent de choix de technologies informatiques et de configuration matérielle.

Elise est un système complexe, comme beaucoup d'outils de text-retrieval. Une bonne partie des performances, tant du point de vue de la satisfaction utilisateur que de l'efficacité algorithmique, se situe "dans les détails".

ANNEXES

1 OKit : documentation

OKit shared library

NAME

OKit - Object Tree, Virtual Machine and Compiler for concatenative languages in a shared library.

AUTHOR

Author :	Yann LANDRIN-SCHWEITZER
Contact :	varkhan@free.fr
Homepage :	http ://varkhan.free.fr/

DESCRIPTION

This library provides an implementation of many basic data types, as integers, floats, strings, lists, as well as executable primitives, encapsulated in a generic object structure. OKit_Lists can contain themselves, as any of these primitive types. Also included are methods for parsing and executing such lists, and error tracking. The library as a whole provides the necessary framework for

manipulating concatenative code.

The object hierarchy tree is described below, stating object name, content, category (between brackets), and string representation (between quotes).

Object		Generic object	[none]	—
	Void	Undefined object	[null]	'¬'
	Intg	Integer number on 32 bits	[scal]	'123456'
	Fltp	Floating point number	[scal]	'3.1415926e+00'
	Strg	Character string	[scal]	'"any text"'
	List	All-purposes list	[list]	'[obj1 obj2 ...]'
	Code	Dynamic executable code	[exec]	'<CODE_NAME>' or '<CODE_NAME :data>'

Note 1 : objects in an OKit_List can be separated by spaces, tabs, or linefeeds / carry returns

Note 2 : here is a table of numeric values for special characters :

CHR	DEC	HEX	CHR	DEC	HEX
"	34	0x22	:	58	0x3A
<	60	0x3C	>	62	0x3E
[	91	0x5B	]	93	0x5D
¬	172	0xAC			

Execution

"*OKit_ListEval*" or "*OKit_ObjectEval*" respectively take a **OKit_List** or an **OKit_Object** in addition to a **OKit_Stack** and **OKit_NameSpace**, on which they will execute the **OKit_List** or **OKit_Object**.

The **OKit_List** type can contain any particular **OKit_Object**. It also provides a mecanism to build a program, since when a **OKit_List** gets executed, all its elements are executed in turn. All **OKit_Code** executable instructions, as well as the execution of a **OKit_List**, use a **OKit_Stack** and a **OKit_NameSpace**. They take arguments and return results on the **OKit_Stack**, while having access to named variables in the **OKit_NameSpace**. Execution of a **OKit_List** is the act to take every member of this list and deposit it on the stack if it is a **OKit_Void**, **OKit_Intg**, **OKit_Strg**, or **OKit_List**, or execute it if it is a **OKit_Code**. Execution stops when the end of the list is reached, or the execution of an [exec] type failed, from lack of arguments, bad argument types, or some other error.

These execution routines return an error status (an **OKit_Status** structure), that is an error code and the associated message, providing a brief description of the error. <0,""> stands for "no error".

Compiler

"*OKit_ListParse*" or "*OKit_ObjectParse*" respectively take a character string and produce a **OKit_List** or an **OKit_Object**. The string is interpretated into objects so that printing these objects would produce back the original string, or something very similar.

Syntax

A valid string for "*OKit_ListParse*" is one or more string representation of objects, among :

Built-ins :	e.g.	' «HELP» ', a builtin command name
Numbers :	e.g.	' 3.1425e+00 ', an integer or floating point number
Strings :	e.g.	' "HELLO" ', a litteral string
OKit_Lists :	e.g.	' [obj1 obj2] ', a list of other objects

During parsing, a bareword is looked-up as a number, then if not applicable in the set of built-in commands, and finally parsed as a string if it could not be found.

Data

The memory representation of an **OKit_Object** has two parts : a generic memory and tracking structure, used for reference counting and cross-referencing, and a content part, type-dependant.

The **OKit_Void** data type is the default type : it is only used to mark internal **OKit_Object** use, or in a standard context that something has gone wrong in memory management.

The three scalar data types are simple counted memory segments : fixed size for the **OKit_Intg** and **OKit_Fltp** types, variable size (which means keeping a '*size*' register) for the **OKit_Strg** type.

The **OKit_List** type is a doubly-linked list, each of its nodes (possibly non-existant) containing an **OKit_Object**.

As for the **OKit_Code** (executable) type, it is a complex structure, containing, among other things, a pointer toward a segment of assembler code retaining the actual implementation, input and output arity information, and a formatted description.

The "system" types (**OKit_Stack** and **OKit_NameSpace**, or named variables table) play an essential role in memory management : **OKit_Objects** can be freed when no reference for them in one of these tables exist any more. They also are the essential holders for instruction arguments and results.

BUGS

Innumerable. Don't forget to report them, even if each bug correction is the source for new ones...

TODO

Implement pseudo-code (list-compiled code) file dumping, in a cross-platform format.

SEE ALSO

OKeval : a library defining a standard built-ins set, and an executable wrapper, enabeling the execution of arbitrary OKit code using these built-ins.

GOKe : a Gtk interface for interactively executing OKit code and manipulating stack and variable lists.

LICENSE

ANY WARRANTY ; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library ; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

2 OKit : formalisation mathématique

2.1 Modélisation du langage

Pour facilliter la modélisation de l'exécution d'un programme OKit, posons quelques définitions. Fixons tout d'abord un ensemble X, de contenus – effectifs – d'objets (on pourra prendre par exemple l'ensemble des chaines binaires, mais la réalité de cet ensemble n'intervient dans aucun des résultats que nous allons énoncer).

▷**Définition 5.1** : **Types**
L'ensemble des **types** $\mathcal{T}$ est un ensemble de trois éléments $\{\mathbf{scal}, \mathbf{list}, \mathbf{inst}\}$

▷**Définition 5.2** : **Prototypes**
Un **prototype** basé sur $\mathcal{T}$ est un couple (p, t) avec $p \in \mathbb{N}$ et $t \in \mathcal{T}^p$. On notera $\mathcal{P}(\mathcal{T})$ l'ensemble de ces prototypes, et τ le prototype "vide" $(0, \{\})$.
Un prototype (p_1, t_1) est inclus dans un prototype (p_2, t_2) si $p_1 < p_2$ et $\forall i \in [[1, p]], t_1^i = t_2^i$

▷**Définition 5.3** : **Objets**
Un **objet** est un quadruplet (t, e, s, u), avec $t \in \mathcal{T}$, e et s deux ensembles finis d'éléments de $\mathcal{P}(\mathcal{T})$, tels que :
- Si $t = \mathbf{scal}$ alors $e = \{\tau\}$, $s = \{\mathbf{scal}\}$, et $u \in X$.
- Si $t = \mathbf{list}$ alors $e = \{\tau\}$, $s = \{\mathbf{list}\}$, et $\exists n \in \mathbb{N} \mid u \in O^n$.
- Si $t = \mathbf{inst}$ alors u est une fonction sur et à valeurs dans l'ensemble des couples (k, a), tels que $k \in \mathbb{N}$ et $a \in O^n$.

On dira que t est le type de l'objet. e et s, respectivement seront appelés ses profils d'entrée et de sortie.

▷**Définition 5.4** : **Atomes d'un objet**
Pour un objet $o = (t, e, s, u)$, on définit l'ensemble $\alpha(o) \subset X$ des atomes de o, par :
- Si $t = \mathbf{scal}$, $\alpha(o) = u$.
- Si $t = \mathbf{list}$, par définition $\exists n \in \mathbb{N} \mid u \in O^n$, et $\alpha(o) = \cup_{i=1}^{n} \alpha(u_i)$.
- Si $t = \mathbf{inst}$, $\alpha(o) = \emptyset$.

▷**Définition 5.5** : **Pile, instance d'un prototype et inclusion**
Une pile est un couple (k, a), tel que $k \in \mathbb{N}$ et $a \in O^k$.
Une pile (k, a) est une instance d'un prototype (p, t) si $k = p$ et $\forall i \in [[1..p]]$, o_i s'écrit (t_i, e, s, u).

Une pile (k_1, a_1) est incluse dans une autre pile (k_2, a_2) si $k_1 \leq k_2$ et $\forall i \in [[1..k_1]], a_1^i = a_2^i$.

Notons que toute pile est une instance du prototype vide τ.

▷**Définition 5.6** : **Objets valides**
On note $\mathcal{O}$ l'ensemble des objets, dit "valides", $o = (t, e, s, u)$ vérifiant :
- Si $t = \mathbf{list}$, $\alpha(o)$ est fini.
- Si $t = \mathbf{inst}$, pour toute instance i d'un prototype dans e, pour toute pile j contenant i, $u(j)$ est défini et contient $u(i)$, et $u(i)$ est une instance d'un prototype de s.

2.2 Propriétés des prototypes

Nous avons déja (au cours de la construction du modèle d'objet) défini les notions de profil d'entrée et de sortie. Nous allons voir qu'il est possible d'étendre ce concept à une suite d'objets dans son ensemble.

▷**Définition 5.7** : **Compatibilité**
Nous dirons que deux prototypes ont compatibles si l'un est inclus dans l'autre, ou l'inverse.

▷**Définition 5.8** : **Composabilité**
Soit deux objets $o_1 = (t_1, e_1, s_1, u_1)$ et $o_2 = (t_1, e_1, s_1, u_1)$. Nous dirons que o_1 est composable avec o_2 si il existe dans s_1 et dans e_2 deux prototypes compatibles.

▷**Définition 5.9** : **Profil de composition**
Soit deux objets $o_1 = (t_1, e_1, s_1, u_1)$ et $o_2 = (t_1, e_1, s_1, u_1)$ composables. Pour chaque $(p_{1,e}, t_{1,e}) \in e_1$, $(p_{1,s}, t_{1,s}) \in s_1$, $(p_{2,e}, t_{2,e}) \in e_2$, $(p_{2,s}, t_{2,s}) \in s_2$, avec $(p_{1,s}, t_{1,s})$ et $(p_{2,e}, t_{2,e})$ compatibles, nous construisons (p_e, t_e) et (p_s, t_s) tels que :
- Si $(p_{1,s}, t_{1,s})$ est inclus dans $(p_{2,e}, t_{2,e})$,
  - $p_e = p_{1,e} + p_{2,e} - p_{1,s}$, et $t_e^i = t_{1,e}^i$ si $i \leq p_{1,e}$, $t_{1,e}^i$ sinon
  - $p_s = p_{2,s}$, $t_s = t_{2,s}$
- Si $(p_{2,e}, t_{2,e})$ est inclus dans $(p_{1,s}, t_{1,s})$,
  - $p_e = p_{1,e}$, $t_e = t_{1,e}$
  - $p_s = p_{2,s} + p_{1,s} - p_{2_e}$, et $t_s^i = t_{2,s}^i$ si $i \leq p_{2,s}$, $t_{1,s}$ sinon

L'ensemble des (p_e, t_e), et celui des (p_s, t_s) forment respectivement les profils de composition d'entrée et de sortie de o_1, o_2, notés $\bar{p}_e(o_1 \bullet o_2)$ et $\bar{p}_s(o_1 \bullet o_2)$.

Il est possible de définir de la même manière le profil de composition de trois objets, en remplaçant dans la définition précédente les profils d'entrée et de sortie du premier objet par les profils de composition des deux premiers, ou en remplaçant les profils du dernier par les profils de composition des deux derniers. Nous noterons les profils obtenus dans le premier cas $\bar{p}_{\{e,s\}}((o_1 \bullet o_2) \bullet o_3)$, dans le second $\bar{p}_{\{e,s\}}(o_1 \bullet (o_2 \bullet o_3))$.

▷**Propriété 5.1** : **Associativité des profils de composition**
Pour tous objets o_1, o_2, o_3 :

$$\bar{p}_e((o_1 \bullet o_2) \bullet o_3) = \bar{p}_e(o_1 \bullet (o_2 \bullet o_3))$$
$$\bar{p}_s((o_1 \bullet o_2) \bullet o_3) = \bar{p}_s(o_1 \bullet (o_2 \bullet o_3))$$

Cette propriété permet de définir la notion de *profil d'une suite d'objets* $(o_1, ..., o_n)$, comme le profil de composition

$$\bar{p}_{\{e,s\}}(o_1 \bullet (o_2 \bullet (...)))$$

et fournit une méthode de calcul itérative, permettant de mettre en place l'algorithme de calcul et de vérification des profils d'entrée et de sortie.

3 OKit : tests de performance

3.1 Principe du test : nombre de cycles par instruction

Evaluer la performance d'un langage "atomique" (composé d'une suite d'instructions indépendantes) peut se faire en comparant les temps d'exécution d'algorithmes simples, qui peuvent s'écrire tant dans ce lanage qu'en langage machine (assembleur). L'évaluation de performance peut alors être exprimée comme le nombre moyen de cycles nécessaire à l'exécution d'une instruction du langage.

Il s'agit d'une mesure empirique, qui peut dépendre assez fortement des algorithmes utilisés comme référence. En effet, une telle moyenne dépend de la "composition" de l'ensemble d'instructions utilisé au sein de l'algorithme. La performance des instructions n'est en effet pas uniforme. Si les instructions les plus efficaces sont aussi les plus souvent appelées, la performance sera surévaluée.

Cette évaluation intègre deux composantes : la performance de chaque instruction utilisée, et ce que l'on peut appeler la "performance du langage", qui traduit le coût de gestion du flux d'exécution et des flux de données. L'ensemble d'instructions étant indépendant de la structure de base de OKit, il est possible de mesurer presque directement cette deuxième composante en créant un ensemble d'instruction *ad-hoc*, pour lequel le coût individuel de chaque instruction est très faible. Par exemple, des instructions OKit réalisant une seule instruction du processeur : calculs et tests arithmétiques, stockage de données.

L'évaluation proprement dite consiste à faire fonctionner l'algorithme de test dans sa version OKit et dans sa version assembleur le même nombre de fois (un nombre élevé permet d'améliorer la précision de la mesure). Le ratio des temps d'exécution totaux des deux versions donne le nombre de cycles processeur nécessaire en moyenne pour exécuter une instruction OKit.

Si l'algorithme de test est simple, on peut aussi se contenter d'évaluer directement la complexité de la version assembleur. En adoptant l'hypothèse simplificatrice que le processeur exécute exactement une instruction assembleur par cycle, on obtient une valeur similaire à la précédente en calculant le ratio $\frac{T.S}{n}$, où T est le temps d'exécution de la version OKit, S la fréquence du processeur (MHz), et n le nombre d'instruction assembleur équivalent, provenant de l'étude de complexité.

3.2 Tests arithmétiques

Les algorithmes de factorisation de nombres, par leur complexité et leur utilisation assez comlète de toutes les instructions arithmétiques, sont très souvent utilisés comme moyen d'évaluation et de comparaison des performances des processeurs eux-mêmes.

Le coût intrinsèque d'exécution d'instrucions OKit arithmétiques est faible, ce qui permet de laisser apparaître clairement la performance du langage OKit lui-même, si l'on utilise ce type d'algorithme dans les tests.

Une version particulièrement intéressante est le calcul de nombres premiers, par le crible d'Eratostène. Il s'agit d'un algorithme simple, mais d'exécution assez coûteuse même pour des quantités de nombres permiers recherchées assez petites. Mais cet algorithme réalise un bon mélange d'instruction arithmétiques, de stockage et de recherche de données. C'est donc un bon candidat pour permettre une évaluation de performance sans biais trop marqué.

BIBLIOGRAPHIE

[Aga97] A. Agapie, "Genetic algorithms : Minimal conditions for convergence" in AE'97, *3rd International Conference on Artificial Evolution*, October 1997, Nîmes, France. Springer Verlag, 1997.

[Alt94] Lee Altenberg, "The Schema Theorem and Price's Theorem" in *Foundation of Genetic Algorithms 3*, 1995. D. Whitley, M. Vose (Eds.), pp. 23-49. Morgan Kaufmann, San Francisco.

[Ban01] W. Banzhaf, "Interactive Evolution" in *Handbook of Evolutionary Computation*, 1997, Oxford University Press.

[BenSch00] S. Ben Hamida, M. Schoenauer, "An Adaptive Algorithm for Constrained optimization Problems" in PPSN VI, *Parallel Problem Solving from Nature VI*, Proceedings of PPSN 4, LNCS 1917, Springer Verlag 2000.

[Bou01] A. Boumaza and J. Louchet, "Dynamic Flies : Using Real-Time Parisian Evolution in Robotics" in *EVOIASP 2001*, Lake Como, Italy, 2001.

[Bra77] R. J. Brachman "What's in a Concept : Structural Foundations for Semantic Networks" in *International Journal of Human-Computer Studies* 9, 1977

[Bro77] C. Brouard "Construction et exploitation de réseaux sémantiques flous pour l'extraction d'information pertinente : le système RELIEFS," Thèse de Doctorat, LIP6, Université Paris-6, 2000

[BucBau+99] A. G. Büchner, M. Baumgarten, M. Mulvenna, S. Anand, J. Hughes, "Navigation Pattern Discovery from Internet data" in *ACM Worksop on Web Usage Analysis and User Profiling*, 1999.

[@CGI] "Common Gateway Interface specification" *W3C Working Draft* `http ://hoohoo.ncsa.uiuc.edu/cgi/interface.html`

[CanKam01] E. Cantu-Paz, C. Kamath, "On the use of Evolutionary Algorithms in Data Mining" in *Data Mining : a Heuristic Approach*, H. A. Abbass, R. A. Sarker and C. S. Newton (Eds), Idea Group Publishing, 2001.

[CanKam00] E. Cantu-Paz and C. Kamath, "On the use of evolutionary algorithms in data mining" in *Data Mining : a Heuristic Approach*, Abbass, H., Sarker, R. and Newton, C. (Eds.) IDEA Group Publishing, 2002. pp. 48-71.

[Cer95] R. Cerf, "Asymptotic convergence of genetic algorithms", in AE'95, *2nd International Conference on Artificial Evolution*, September 1995, Brest, France - LNCS 1063, pp 37-54, Springer Verlag, 1995.

[ChaLut01] J. Chapuis, E. Lutton "ArtiE-Fract : Interactive Evolution of Fractals" in *GA '01, Generative Art Conference*, Milano, Italy, 2001

[Che95] H. Chen, "Machine learning for information retrieval : neural networks, symbolic learning and genetic algorithms" JASIS vol. 46 (3), *Journal of the American Society for Information Science and Technology*, April 1995.

[CheSha+98] H. Chen, G. Shankaranarayanan, L. She and A. Iyer, "A Machine Learning Approach to Inductive Query by Examples : An Experiment Using Relevance Feedback, ID3, Genetic Algorithms and Simulated Annealing" JASIS vol. 49 (8), *Journal of the American Society for Information Science and Technology*, June 1998.

[CleKee66] C. W. Cleverdon, E. M. Keen. "Factors determining the performance of indexing systems" *Aslib Cranfield Research Project*, 1966, Cranfield, England.

[Cod70] E. F. Codd "A relational model for large shared data banks" in ACM'70, *Communications of the ACM*, Vol. 13(6), pp 377-387, June 1970

[ColQui72] A. M. Collins, M. R Quillian *Experiments on semantic memory and language comprehension* in *Cognition in learning and memory.* L W Gregg (Ed), New York : Wiley, 1972

[Dav89] L. Davis, "Adapting operator probabilities in Genetic Algorithms" in ICGA '89, *International Conference on Genetic Algorithms and their Application*, 1989.

[TDav91] T. E. Davis and J. C. Principe. "A Simulated-Annealing-Like Convergence Theory for the Simple Genetic Algorithm" in *4th International Conference on Genetic Algorithm*, July 13-16 1991. pp 174-182.

[DeJSpe0] K. A. DeJong, W. M. Spears, "An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms" in *First Workshop Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1990. pp. 38-47.

[DeeDum+88] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, "Using latent semantic analysis to improve information retrieval" in CHI'88, *Conference on Human Factors in Computing*, 1988, New York : ACM. pp 281-285.

[DeeDum+90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman, "Indexing by Latent Semantic Analysis" in JASIS, *Journal of the American Society for Information Science*, vol 41 (6), 1990, pp 391-407.

[Die00] R. Diestel "Graph Theory" *Graduate Texts in Mathematics, Volume 173* Springer-Verlag, New York, May 1997

[Dix98] A. Dix, "Interactive Querying, locating and dicovering information" in *Second Workshop on Information Retrieval and Human Computer Interaction*, Glasgow, 11th September 1998. `http ://www.hiraeth.com/alan/topics/QbB/`

[EveRot+99] Y. Even-Zohar, D. Roth, D. Zelenko "Word prediction and clustering" in *Bar-Ilan Symposium on the Foundations of Artificial Intelligence*, Ramat-Gan, Israel, 1999

[EynJak+00] F. Van Eynde, Z. Jakub, D. Walter, "Part of Speech Tagging and Lemmatisation for the Spoken Dutch Corpus" LREC 2000, *Second International Conference on Language Resources & Evaluation*

[FelMil98] C. Fellbaum, G. A. Miller, "WordNet : An Electronic Lexical Database for the English Language," The MIT Press, May 1998

[FieRog00] C.-N. Fiechter, S. Rogers "Learning Subjective Functions with Large Margins" in ICML'00 *17th International Conference on Machine Learning*, 2000 Proceedings pp 287-294.

[Fos97] D. J. Foskett, "Thesaurus" in *Readings in Information Retrieval*, K. S. Jones, P. Willet, M. Kaufmann Publishers, San Fransisco, 1997

[Fra98] O. François, "An Evolutionary Strategy for Global Minimization and its Markov Chain Analysis" in *IEEE Transactions on Evolutionary Computation*, September 1998.

[Fre99] A.A. Freitas, "Data Mining with Evolutionary Algorithms : Research Directions" *AAAI Worshop, Technical Report WS-99-06*, ISBN 1-57735-090-1, The AAAI Press, 1999.

[Fre01] A. Freitas, "A survey of evolutionary algorithms for data mining and knowledge discovery" in *Advances in evolutionary computation*, A. Ghosh, S. Tsutsui (Eds.), Springer-Verlag, 2001

[Fre02] A.A. Freitas, "A survey of evolutionary algorithms for data mining and knowledge discovery" in *Advances in Evolutionary Computation*, A. Ghosh, S. Tsutsui. (Eds.), Springer-Verlag, 2002.

[Gol89] D. A. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* Addison-Wesley Publishing, 1989.

[Gor88] M. D. Gordon, "Probabilistic and Genetic Algrithms for Document Retrieval" in *Communications of the ACM* 31, pp 1208-1218, 1988.

[Gre86] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms" in *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, Jan./Feb. 1986, pp. 122-128.

[Gru93] T. R. Gruber "A Translation Approach to Portable Ontology Specifications" Academic Press. June 1993.

[HagSte99] J. Hagman, R. Steinberger, D. Perrotta, A. Varfis "Approaches to Document Classification and Visualisation" in IJCAI'99 *International Joint Conferences on Artificial Intelligence 1999*, Stockholm, Sweden, July 31 - August 6, 1999.

[HagSte00] J. Hagman, D. Perrotta, R. Steinberger, A. Varfis "Document Classification and Visualisation to Support the Investigation of Suspected Fraud" in PKDD'00 *4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Lyon, France, September 13-16, 2000.

[HarLob99] G. Harik, F. Lobo, "A parameter-less genetic algorithm" in *IlliGAL Technical Report 99009*, Jan 1999.

[Hea99] M. Hearst, "Untangling Text Data Mining" ACL'99 *37th Annual Meeting of the Association for Computational Linguistics*, University of Maryland, June 20-26, 1999

[Hof99] T. Hofmann, "Probabilistic Latent Semantic Indexing" in SIGIR '99, *International ACM SIGIR Conference on Research and Development in Information Retrieval*, Berkeley, USA, 1999.

[Hor97] J. Horn, "The nature of niching : genetic algorithms and the evolution of optimal, cooperative populations." IlliGAL Report no 97008, 1997, University of Illinois, Champaigne, Illinois, USA.

[HorYeh00] J.-T. Horng, C.-C. Yeh, "Applying Genetic Algorithms to Query Optimisation in Document Retrieval" in *Information Processing and Management 36*, 2000, pp 737-759.

[KamTak+97] S. Kamohara, H. Takagi, T. Takeda, "Control Rule Acquisition for an Arm Wrestling Robot" in SMC'97 (vol 5), *IEEE Int. Conf. on System, Man and Cybernetics* Orlando, FL, USA, 1997, pp 4227-4231.

[KimZha+00] Y.-H. Kim, S. Kim, J.-H. Eom and B.-T. Zhang, "SCAI Experiments on TREC-9" in TREC-9, *Ninth Text REtrieval Conference*, 2000, pp. 392-399.

[KimZha01] S. Kim and B.-T. Zhang, "Evolutionary Learning of Web-Document Structure for Information Retrieval" in CEC'01 *2001 Congress on Evolutionary Computation*, vol. 2, pp. 1253-1260.

[Kin99] B. Kindt, "Recherches de lexicographie grecque à l'Institut orientaliste" in *8ème Bulletin de liaison du Département d'Études grecques, latines et orientales*, septembre 1999,

[Koz92] J. R. Koza, "Symbolic Regression - Error-Driven Evolution" in *Genetic Programming I : On the Programming of Computers by Means of Natural Selection*, J. R. Koza, MIT Press, Cambridge, Massachusetts, 1992.

[LabMon+03a] N. Labroche, N. Monmarche, G. Venturini, "Web Sessions Clustering with Artificial Ant Colonies" in WWW'03, *World Wide Web Conference 2003*, Budapest, Hungary, May 20-24 2003.

[LabMon+03b] N. Labroche, N. Monmarche, G. Venturini, "Antclust : Ant Clustering and Web Usage Mining" in GECCO'03, *Genetic and Evolutionary COmputation Conference*, Chicago, USA, July 12-16 2003.

[YLSLut00] Y. Landrin-Schweitzer, E. Lutton, "Perturbation theory for Evolutionary Algorithms" in PPSN VI, *Parallel Problem Solving*

from Nature VI, Proceedings of PPSN 4, LNCS 1917, Springer Verlag 2000.

[LanPol97] W. B. Langdon, R. Poli, "Fitness causes bloat" in *Soft Computing in Engineering Design and Manufacturing*, P.K Chawdhry et al. (Eds.), Springer Verlag 1997.

[LeGDod01] B. Le Grand, M. Soto, D. Dodds "XML Topic Maps and Semantic Web Mining" in *XML Conference 2001* Orlando, Florida, USA. December 9-14, 2001

[LebLut97] B. Leblanc, E. Lutton, *Bitwise Regularity Coefficients as a Tool for Deception Analysis of a Genetic Algorithm*, 1997, INRIA research report, RR-3274.

[LebLut+98] B. Leblanc, E. Lutton, B. Braunschweig, H. Toulhoat, "History and Immortality in Evolutionary Computation", in EA'01 *5th International Conference on Artificial Evolution*, 2001.

[LeuChe+00] M. L. Wong, K. S. Leung and J. C.Y. Cheng, "Discovering Knowledge from Noisy Databases Using Genetic Programming" JASIS vol. 51 (9), *Journal of the American Society for Information Science and Technology*, July 2000.

[Li92] Wentian Li "Random Texts Exhibit Zipf's Law-Like Word Frequency Distribution" in IEEETIT (38) *IEEE Transactions on Information Theory*, 1992.

[Los01] R. M. Losee "Term dependence : A basis for Luhn and Zipf models" in JASIS 12(52) *Journal of the American Society of Information Science*, 2001. pp 1019-1025.

[LutCol02] P. Collet, J. Louchet, E. Lutton, "Issues on the Optimisation of Evolutionary Algorithms Code" in CEC'02 *Congress on Evolutionary Computation*, Honolulu, USA, May 12-17, 2002.

[Mah97] S. W. Mahfoud, "Niching Methods for Genetic Algorithms" PhD Thesis, 1995, University of Illinois, Champaigne, Illinois, USA.

[MasTuf97] Oliver Mason, Dan Tufiș, "Probabilistic Tagging in a Multi-lingual Environment : Making an English Tagger Understand Romanian" in *3rd TELRI European Conference*, Montecatini, Italy, Oct.1997

[MNa66] R. McNaughton, "Testing and generating infinite sequences by a finite automaton" in *Information and Control*, vol 9, 1966, pp 521-530.

[MNa74] R. McNaughton, "Algebraic decision procedures for local testability" in *Math. Systems Theory*, vol 8, 1974, number 1, pp 60-76.

[MNaPap71] R. McNaughton, S. Papert, "Counter-free automata" in *M.I.T. Research Monograph No 65*, The M.I.T. Press, Cambridge, Mass.-London, 1971.

[MeyGan+97] C. Meyer, J.-G. Ganascia, J.-D. Zucker, "Learning Strategies in Games by Anticipation," in IJCAI '97, *International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997

[MonVen+99] N. Monmarche, G. Nocent, G. Venturini, P. Santini. "On Generating HTML Style Sheets with an Interactive Genetic Algorithm Based on Gene Frequencies" in AE'99, *4th International Conference on Artificial Evolution*, Dunkerque, France, November 1999, C. Fonlupt, J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer (Eds), Springer Verlag, LNCS 1829.

[NeiRya98a] M. O'Neill, C. Ryan, "Grammatical Evolution : A Steady State approach" in *Proceedings of the Second International Workshop on Frontiers in Evolutionary Algorithms*, 1998, pp 419-423.

[NeiRya+98b] C. Ryan, J. J. Collins, M. O'Neill, "Grammatical Evolution : Evolving Programs for an Arbitrary Language" in *First European Workshop on Genetic Programming*, 1998, LNCS 1391, Springer-Verlag.

[PalTal+02] S. Pal, V. Talwar, P. Mitra, "Web Mining in Soft Computing Framework : Relevance, State of the Art and Future Directions," in *IEEE Transactions on Neural Networks, 2002* `http ://citeseer.nj.nec.com/pal02web.html`

[Par98] F. Parmentier, "Spécification d'une architecture émergente fondée sur le raisonnement par analogie : Application aux références bibliographiques" Thèse de Doctorat, juin 1998, Université Henri-Poincaré - Nancy 1

[ParFre+02] R.S. Parpinelli, H.S. Lopes, A.A. Freitas, "Data Mining with an Ant Colony Optimization Algorithm" in *IEEE Transactions on Evolutionary Computation*, special issue on Ant Colony algorithms. 2002.

[PolCag97] R. Poli, S. Cagnoni, "Genetic Programming with User-Driven Selection : Experiments on the Evolution of Algorithms for Image Enhancement" in *2nd Annual Conf. on Genetic Programming*, 1997 pp 269-277.

[Qui00] Aaron J. Quigley "Large Scale 3D Clustering and Abstraction" in VIP'00, *Visual Information Processing 2000*, Sydney December 1, 2000.

[Qui86] J. R. Quinlan, "Induction of Decision Trees" in *Machine Learning, no 1*, 1986.

[RatSeb00] A. Ratle, M. Sebag, "Genetic Programming and Domain Knowledge : Beyond the Limitations of Grammar-Guided Machine Discovery" in PPSN 2000, *Parallel Problem Solving from Nature VI*, September 2000, Paris, France.

[RayLut+00] P. Collet, E. Lutton, F. Raynal, M. Schoenauer, "Polar IFS + Parisian Genetic Programming = Efficient IFS Inverse Problem Solving" in *Genetic Programming and Evolvable Machines Journal*, vol. 1 (4), October, 2000. pp. 339-361

[Rud97] G. Rudolph, "Asymptotical convergence rates of simple evolutionary algorithms under factorizing mutation distributions" in AE'97, *3rd International Conference on Artificial Evolution* Nîmes, October 1997, France. Springer Verlag, 1997.

[SalMcG83] G. Salton, M. J. McGill *Introduction to Modern Information Retrieval*, McGraw-Hill, January 1983.

[SchMor87] J. D. Schaffer, A Morishima, "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms" in ICGA '87, *International Conference on Genetic Algorithms*, 1987.

[SebRav+96] M. Sebag, C. Ravisé, M. Schoenauer, "Controlling Evolution by Means of Machine Learning" in EP'96, *5th Annual Conference on Evolutionary Programming*, March 1996, San Diego, USA.

[SEM5] "Search Engines Today and the New Frontier" *The Fifth Search Engine Meeting* Boston, Massachusetts, April 10-11, 2000.

[SEM8] "The Synergy between Search, Systems and Information" *The Eighth Search Engine Meeting*, Boston, Massachusetts, April 7-8, 2003.

[ShiKim+99] D.-H. Shin , Y.-H. Kim, S. Kim, J.-H. Eom, H.-J. Shin and B.-T. Zhang, "SCAI TREC-8 Experiments" in TREC-8, *Eighth Text Retrieval Conference*, 1999, pp. 511-518.

[Sim91a] K. Sims, "Artificial Evolution for Computer Graphics" in *Computer Graphics*, vol 25 (4), July 1991, pp 319-328.

[Sim91b] K. Sims, "Interactive evolution of dynamical systems" in *First European Conference on Artificial Life*, Paris, December 1991, pages 171-178.

[ArrSmi98] P. Smith, K. Arries, "Code growth, explicitely defined introns, and alternative selection schemes" in CEC'98, *1998 Congress on Evolutionary Computation*.

[SolCra98] S. Solstysiak, B. Crabtree, "Automatic Learning of User Profiles - Towards the Personalisation of Agent Service" in *BT Technology Journal* vol 16 (3), July 1998.

[Spe91] W. M. Spears, "Adapting crossover in a Genetic Algortithm" in ICGA '91, *International Conference on Genetic Algorithms*, 1991.

[Tak98] H. Takagi, " Interactive Evolutionary Computation : System Optimisation Based on Human Subjective Evaluation" in INES'98, *IEEE International Conference on Intelligent Engineering Systems*, Vienna, Austria, September 17-19 1998, pp 1-6.

[TakOhs99] H. Takagi, M. Ohsaki, "IEC-based Hearing Aids Fitting" in SMC'99 *IEEE International Conference on System, Man and Cybernetics*, Tokyo, Japan, October 12-15 1999, vol 3, pp 657-662.

[TodLat92] S. J. P. Todd, W. Latham, *Evolutionary Art and Computers* Academic Press, 1992.

[Tro93] A. Trouvé, *Rough large deviation estimates for the optimal convergence speed exponent of generalized simulated annealing algorithms.* 1993

[TseYan98] L.Y. Tseng, S. B. Yang, "Genetic Algrithms for Clustering. Feature selection and Classification" in *International Conference on Neural Networks*, 1997, vol 3, pp 1612-1616.

[Tur03] P. D. Turney "Coherent keyphrase extraction via Web mining" IJCAI'03, *The Eighteenth International Joint Conference on Artificial Intelligence* Acapulco, Mexico, 2003, pp 434-439.

[VacPar+01] T. Vachon, N. Grandjean, P. Parisot, "Interactive Exploration of Patent Data for Competitive Intelligence : Applications in Ulix (Novartis Knowledge Miner)" *International Chemical Information Conference and Exhibition*, Nîmes, France, October 21-24 2001.
`http ://www.infonortics.com/chemical/ch01/01chempro.html`

[VilLar+99] M. J. Martin-Bautista, M.-A. Vila, H. L. Larsen, "A Fuzzy Genetic Algorithm Approach to an Adaptive Information Retrieval Agent" JASIS vol 50 (9), *Journal of the American Society for Information Science and Technology*, July 1999.

[Voo94] Ellen M. Voorhees, "Query expansion using lexical-semantic relations" in *17th Annual International ACM/SIGIR Conference*, Dublin, Ireland, 1994.

[Vos90] M.D. Vose, "Formalizing genetic algorithms" in *Genetic Algorithms, Neural Networks and Simulated Annealing Applied to Problems in Signal and Image processing*. Kelvin Conference Centre, University of Glasgow, May 8-9, 1990. IEEE.

[Vra97] D. Vrajitoru, "Genetic Algorithms in Information Retrieval" in *AIDRI97*, Learning; From Natural Principles to Artificial Methods, Genève, June 1997.

[Vra00] D. Vrajitoru, "Large Population or Many Generations for Genetic Algorithms ? Implications in Information Retrieval" in *Soft Computing in Information Retrieval. Techniques and Applications* F. Crestani, G. Pasi (Eds.) Physica-Verlag, Heidelberg, 2000, pp 199-222.

[WalSmi00] N. Walton, G. D. Smith, "The Origination of Diversity by Adaptive Clustering" in PPSN VI, *Parallel Problem Solving from Nature VI*, Proceedings of PPSN 4, LNCS 1917, Springer Verlag 2000.

[Win99] W. Winiwarter, "PEA-A Personal E-mail Assistant with Evolutionary Adaptation" *International Journal of Information Technology*, 1999.
`http ://citeseer.nj.nec.com/winiwarter99pea.html`

[YanRas+93] J. Yang, R. R. Korfhage, E. Rasmussen, "Query Improvement in Information Retrieval using Genetic Algorithms : A Report on the Experiments of the TREC project" in TREC-1, *The first Text Retrieval Conference*, 1993.

[YanHon98] J. Yang, V. Honavar, *Feature extraction, Construction and Selection - A Data Mining Perspective* Kluver Academics Publishes, 1998, pp 117-136.

[ZhaKwa+96] B.-T. Zhang, J.-H. Kwak, C.-H. Lee, "Building Software agents for information fitlering on the internet : A Genetic Programing approach" *Genetic Programing Conference*, 1996.

[SGML] "Standard Generalized Markup Language (SGML)" *ISO 8879*, 1986. Information Processing – Text and Office Systems
`http ://www.iso.ch/cate/d16387.html`

[SQL] "Information Technology - Database Language SQL" ISO/IEC specifications 9075-1 :1999, 9075-2 :1999, 9075-3 :1999, 9075-4 :1999, 9075-5 :1999, 9075-9 :2001, and 9075-10 :2000

[HTML-2.0] T. Berners-Lee, D. Connolly, "HyperText Markup Language – HTML 2.0" *RFC no 1866*, November 1995.
`http ://www.w3.org/Protocols/rfc18166/rfc18166.html`

[HTTP-1.1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1" *RFC no. 2616*, June 1999.
`http ://www.w3.org/Protocols/rfc2616/rfc2616.html`

[HTML-4.0] S. Pemberton et al., "Hypertext Markup Language (HTML) 4.01 (First Edition)" *W3C Recommendation*, December 24, 1999. `http ://www.w3.org/TR/html4/`

[XHTML-1.0] S. Pemberton et al., "XHTML^{TM} 1.0 : The Extensible HyperText Markup Language" (W3C Proposed Recommendation) `http ://www.w3.org/TR/xhtml1`

[XML-1.0] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)" *W3C Recommendation*, October 6, 2000.
`http ://www.w3.org/TR/REC-xml`

[@Best] "The BEST Search Engines" UC Berkeley - Teaching Library Internet Workshops
`http ://www.lib.berkeley.edu/TeachingLib/Guides/Internet/SearchEngines.html`

[@OKit] Y. Landrin-Schweitzer, "OKit, a Virtual Machine and Compiler for Concatenative Languages"
`http ://sourceforge.net/projects/libokit`

[@SWISH] SWISH++, Simple Web Indexing System for Humans : C++ version,
`http ://homepage.mac.com/pauljlucas/software/swish/`

[@AV] AltaVista Search Services,
`http ://www.altavista.com/en/corporate.html`

[@AMAZON] Amazon, `http :/www.amazon.com`

[@CFD] Cystic Fibrosis Reference Collection,
`http ://www.sims.berkeley.edu/~hearst/irbook/cfc.html`

[@EMBASE] "EMBASE, the Excerpta Medical database" Elsevier Science, Secondary Publishing Division, New York accessible on-line http ://www.ncbi.com/embase

[@GOOGLE] Google Search, http ://www.google.com

[@INK] Inktomi Search Solutions, http ://www.inktomi.com

[@JAVA] JAVA^{TM} Sun Microsystems http ://java.sun.com

[@MESH] "MeSH (Medical Subject Headings), a controlled vocabulary thesaurus" http ://www.nlm.nih.gov/pubs/factsheets/mesh.html National Library of Medicine (NLM), National Institutes of Health, Bethesda, Maryland

[@PHP] PHP : HyperText Processor, http ://www.php.net

[@TREC] The Text REtrieval Conference (TREC) homepage http ://trec.nist.gov/

[@W3C] The World-Wide-Web Consortium, http ://w3c.org

[@WNET] G. A. Miller, C. Fellbaum, R. Tengi, P. Wakefield, "WordNet, an Electronic Lexical Database for the English language : Cognitive Science Laboratory, Princeton University. http ://www.cogsci.princeton.edu/ wn/"

[@XHTML] "XHTML^{TM} 2.0" a *W3C Working Draft*, January 31, 2003. http ://www.w3c.org/TR/xhtml2

[@XML] "XML, Extensible Markup Language" http ://www.w3.org/XML

[@RDB] XML representation of a relational database http ://www.w3.org/XML/RDB.html

[@YAHOO] Yahoo Search Engine, http ://www.yahoo.com/corporate.html

L'inflation des quantités de documents disponibles sous forme électronique, sur internet et dans les intranets d'entreprises a entrainé au cours des annéees 1990-2000 le développement d'outils de stockage et de gestion des données électroniques à grande échelle. Parmis ceux-ci, les moteurs de recherche textuelle (text-retrieval) ont désormais une place centrale dans le traitement et la diffusion des informations.

Grâce à des outils sémantiques et linguistiques élaborés, les solutions de recherche textuelle sont devenues très efficaces. Cependant, les particularités des utilisateurs, et l'effort que ceux-ci doivent fournir pour interpréter les résultats de leurs recherches, opposent une dernière barrière à l'amélioration des performances. Les approches statistiques, fondées sur des modèles cognitifs des utilisateurs pour interagir au mieux avec eux, ont prouvé leur efficacité dans des situations au contexte sémantique simple. Mais elles ne permettent pas encore aux outils d'extraction textuelle de répondre de manière personnaliséee et adaptable à des demandes complexes.

L'approche développée dans cette thèse est fondée sur une spécialisation du comportement des outils pour chaque utilisateur. Les modèles cognitifs ne suffisent pas à décrire les comportements des utilisateurs de manière assez complète pour déterminer le traitement le plus adapté des requêtes. Partant de cette constatation, nous avons élaboré un modèle des traitements de requêtes possibles, sans hypothèse a-priori sur le comportement des utilisateurs. Le traitement spécifique à chacun d'eux, contenu dans un profil personnalisé, est adapté dynamiquement grâce à l'historique des documents consultés par l'utilisateur à l'issue des requêtes. Il s'agit d'un problème d'optimisation extrêmement complexe.

L'optimisation des profils est réalisée à l'aide d'un algorithme évolutionnaire, qui maximise une mesure empirique de satisfaction de l'utilisateur. La méthode de programmation génétique parisienne que nous avons développée fait évoluer une population de modules, composants élémentaires de règles de transformation des requêtes. Ces règles sont utilisées pour réécrire les requêtes, ensuite transmise à un moteur de recherche standard permettant de former des listes de documents. De manière invisible pour l'utilisateur, l'évaluation des modules provient de l'analyse des consultations de documents, et fournit à l'algorithme évolutionnaire les informations permettant la mise à jour du profil utilisateur.

Un prototype fonctionnel, Elise, implémente ces mécanismes, bien adaptés à une intégration dans le contexte des intranets et des outils d'extraction textuelle. Si la performance d'Elise, liée aux opinions des utilisateurs, est d'évaluation délicate, les résultats obtenus prouvent qu'Elise a des capacités d'adaptation et de créativité, contrairement aux systèmes traditionnels. Le prototype Elise a permis de prouver la faisabilité de l'approche évolutionnaire interactive pour l'extraction textuelle. L'étape suivante d'intégration dans un cadre industriel du prototype ne nécessite plus que l'intégration d'outils sémantiques et d'analyse textuelle plus performants, et l'amélioration de l'efficacité algorithmique de certains composants.