Recherche d'information sur Internet par algorithmes évolutionnaires

Fabien Picarougne

Université François Rabelais Tours

École Doctorale : Santé, Sciences et Technologies

Année Universitaire : 2003-2004

THÈSE POUR OBTENIR LE GRADE DE
DOCTEUR DE L'UNIVERSITÉ DE TOURS

Discipline : **Informatique**

présentée et soutenue publiquement

par :

Fabien Picarougne

le 19 novembre 2004

Recherche d'information sur Internet par algorithmes évolutionnaires

---

Directeur de thèse : Professeur Gilles Venturini

---

jury :

Philippe Preux	*Rapporteur*	Université de Lille 3
Chantal Reynaud	*Examinateur*	Université de Paris 11
Michèle Sebag	*Rapporteur*	CNRS
Gilles Venturini	*Examinateur*	Université de Tours
Christel Vrain	*Examinateur*	Université d'Orléans
Djamel Zighed	*Président*	Université de Lyon 2

Remerciements

Le travail qui est présenté dans ce document a été effectué au sein de l'équipe Réseau et Technologies de l'Information et de la Communication du Laboratoire d'Informatique (EA 2101) de l'Université de Tours dans les locaux du Département Informatique de l'Ecole Polytechnique de l'Université de Tours. Il est l'aboutissement de nombreux mois de travaux et d'efforts qui furent possibles grâce à un certain nombre de personnes que je souhaite aujourd'hui remercier.

Je remercie en premier lieu Michèle Sebag et Philippe Preux pour avoir accepté de consacrer une partie de leur temps à l'examen et à l'évaluation de mes travaux, ainsi que Chantal Reynaud, Christel Vrain et Djamel Zighed pour avoir accepté de participer à ce jury.

Je remercie tout particulièrement Gilles Venturini qui m'a guidé et porté tout au long de ce travail et avec qui j'ai partagé de bonnes aventures. Il m'a donné la possibilité et les moyens d'arriver jusqu'ici et ses conseils m'ont toujours été précieux. Son attention et son goût pour la recherche sont un modèle pour moi.

Je remercie également Élodie, Vincent et Jérôme pour le temps passé à la correction de ce mémoire. Ce travail ne fut pas de tout repos.

Je tiens également à remercier les membres du Laboratoire d'Informatique de l'Université de Tours avec lesquels nous avons eu de bons moments, pour leur bonne humeur et les instants de fou rire.

Enfin, je ne saurais terminer cette liste sans adresser un remerciement particulier à mon amie Hanene Azzag, avec qui j'ai eu le plaisir de partager tout au long de ces trois années le bureau et bien d'autres choses.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

Le volume de données présentes sur Internet est en augmentation extrêmement rapide, et en faisant une moyenne des estimations trouvées dans des articles de presse ou des articles scientifiques, on peut dire que le nombre de documents accessibles est de l'ordre de la dizaine de milliards. Ces documents peuvent être de nature scientifique, industrielle ou commerciale. Pour des entreprises mais aussi pour des particuliers, ces documents peuvent contenir à un moment donné des informations stratégiques nécessaires à la résolution d'un problème. Cette taille colossale et la demande des utilisateurs posent un défi à la communauté scientifique qui doit être en mesure de proposer des outils efficaces de recherche d'information.

Des moteurs de recherche sont donc apparus depuis des dizaines d'années. À partir d'une requête simple, ils parcourent leur mémoire pour trouver le plus rapidement possible les documents correspondant aux souhaits de l'utilisateur. *Google* est certainement la dernière réussite américaine en ce domaine puisqu'il possède en mémoire l'image de plusieurs milliards de documents. Il fournit souvent en réponse à une requête des milliers de documents en un temps inférieur à la seconde. Ce sont là précisément les points forts et les points faibles de ce type de méthode : l'utilisateur est submergé par des milliers de réponses présentées "en vrac", dont une partie ne correspond pas à la requête, une autre correspond à des pages qui n'existent plus, sans parler de nombreux types de documents qui ne sont pas pris en compte dans la mémoire du moteur. Dans bien des cas, l'utilisateur passe un temps assez long à analyser les résultats du moteur, sans garantie de résultats.

Sujet de la thèse

Nous proposons dans cette thèse de développer un moteur de recherche qui va résoudre les difficultés de ses concurrents en partant du principe très général suivant : plutôt que de donner une réponse très rapide et très pauvre en information, il serait envisageable pour de nombreuses requêtes de fournir des résultats beaucoup plus riches pour l'utilisateur mais en un temps plus long. Pour cela, nous allons utiliser des algorithmes évolutionnaires (algorithmes génétiques et fourmis artificielles) et d'optimisation (algorithme tabou) qui vont aller chercher l'information en parallèle. De cette façon, ce nouveau moteur va pouvoir explorer de nouvelles régions d'Internet avec une stratégie efficace sélectionnant les meilleures pages à explorer à un moment donné.

De manière plus détaillée, nous nous proposons :

a) d'améliorer la formulation de la requête en donnant la possibilité à un utilisateur de décrire le type de résultat qu'il désire obtenir à l'aide de plusieurs critères ;

b) de définir quelle stratégie employer puisque l'on donne plus de temps à la machine pour trouver des pages ; nous allons tester des algorithmes génétiques, un algorithme à base de fourmis artificielles et un algorithme tabou et les comparer à un méta-moteur classique. Cela nécessite 1) de coder le problème de recherche d'informations sous la forme d'un problème d'optimisation, 2) de définir des opérateurs permettant d'explorer de nouvelles pages à partir d'une page ou d'un ensemble de pages, 3) de choisir une méta-heuristique qui permette d'équilibrer la recherche entre la compilation des résultats issus de moteurs de recherche classiques (méta-moteur) et l'exploration de nouveaux liens ;

c) de proposer une architecture parallèle d'un point de vue logiciel et matériel. Il s'agit, pour la partie logicielle, d'utiliser des méta-heuristiques parallélisables et de définir quelles parties des algorithmes il est intéressant de paralléliser. En ce qui concerne la partie matérielle, il s'agit de donner la possibilité d'un parallélisme hétérogène, permettant à la fois plusieurs exécutions concurrentes sur une même machine et plusieurs exécutions sur des machines hétérogènes s'inscrivant sur un serveur. Ainsi, un utilisateur d'un réseau d'entreprise ou d'Internet peut proposer sa machine (ressource de calcul et de connexion) pour former une architecture naturellement massivement parallèle ;

d) de présenter visuellement les résultats du moteur. Nous allons enrichir la présentation des résultats en effectuant automatiquement des opérations réalisées manuellement par les utilisateurs dans les moteurs de recherche actuels, comme la classification automatique et visuelle des résultats en fonction des textes qu'ils contiennent.

Organisation du document

Ce rapport s'organise de la façon suivante :

- Le premier chapitre fait un état de l'art des différents éléments qui composent un moteur de recherche. Il aborde toutes les étapes nécessaires à l'aboutissement d'une recherche, de la formulation de la requête, en passant par les algorithmes de recherche utilisés actuellement pour retrouver des documents pertinents sur Internet, jusqu'à la présentation et la visualisation de résultats. Il permet de donner un aperçu au lecteur des éléments nécessaires à mettre en œuvre dans la conception d'un moteur de recherche.
- Le chapitre 2 présente les algorithmes qui seront utilisés pour réaliser le cœur du moteur de recherche. Nous abordons ici des heuristiques utilisées en optimisation comme les algorithmes évolutionnaires et plus précisément les algorithmes génétiques et les algorithmes basés sur le comportement des fourmis, ou encore la recherche tabou. Nous examinons également des exemples d'adaptation de ces heuristiques au Web et à la recherche d'information.

- Dans le troisième chapitre nous construisons une modélisation d'Internet sous forme d'un graphe permettant de considérer le problème de recherche d'information comme un problème d'optimisation. Cette modélisation permet d'élaborer un modèle de recherche d'information générique dans lequel nous pouvons intégrer des algorithmes basés sur les heuristiques détaillées dans le chapitre précédent. L'architecture utilisée permet d'effectuer une parallélisation distribuant l'effort de recherche sur plusieurs entités situées dans un réseau quelconque.
- Le chapitre 4 développe une approche génétique séquentielle de recherche d'information sur Internet : *Geniminer*. Elle permet de démontrer les capacités du modèle conçu et de construire un méta-moteur de recherche d'information.
- Nous proposons dans le chapitre 5 une évolution de l'algorithme précédent que nous avons nommé *GeniminerII*. Cet algorithme résout les problèmes de sous-utilisation des ressources de *Geniminer* en parallélisant la recherche. Il offre également l'occasion d'introduire une nouvelle requête d'interrogation du moteur de recherche, plus riche, permettant d'analyser plus précisément la pertinence des documents.
- Le chapitre 6 introduit deux nouveaux types d'algorithmes de recherche : *Antsearch*, utilisant une modélisation à base de fourmis et *Tabusearch*, inspirée de la recherche tabou. L'utilisation de ces méthodes permet de démontrer la généricité du modèle de recherche d'information établi au chapitre 3.
- Nous comparons les méta-heuristiques ainsi construites dans le chapitre 7 afin de déterminer quelle méthode s'applique le mieux à notre problème.
- Le chapitre 8 aborde le dernier point d'un moteur de recherche : une présentation efficace des résultats. L'ensemble des documents retournés par le moteur de recherche est regroupé par un algorithme de classification basé sur les nuages d'agents. L'analyse des résultats obtenus permet de conclure à l'efficacité d'une telle représentation.
- À la fin de ce document, une conclusion fait le bilan sur l'ensemble de ces travaux et indique les perspectives de développement de nos outils. Il s'agit en particulier de compléter les critères de la requête utilisateur en proposant par exemple de spécifier un ensemble de documents types ; les résultats devront alors être similaires à ces éléments. Il est également possible de renforcer l'aspect de veille stratégique en capitalisant les connaissances extraites de résultats issus de plusieurs requêtes. Il est alors possible de partager l'ensemble des informations récoltées entre un groupe d'utilisateurs soulignant les éléments les plus importants pour la communauté toute entière (filtrage collaboratif). Enfin, l'aspect de visualisation des résultats peut être renforcé en rajoutant de l'interactivité par une hybridation de notre méthode avec un algorithme d'affichage de graphes à base de forces et de ressorts.

Ce travail de thèse a fait l'objet de quatre publications dans des conférences nationales [Picarougne *et al.*, 2002a], [Picarougne *et al.*, 2003a], [Picarougne *et al.*, 2003b] et [Picarougne *et al.*, 2004b] et de cinq publications dans des conférences internationales [Picarougne *et al.*, 2002b], [Picarougne *et al.*, 2002c], [Picarougne *et al.*, 2002d], [Picarougne *et al.*, 2003c] et [Picarougne *et al.*, 2004a].

Chapitre 1

La recherche d'information sur Internet

Résumé

Ce chapitre aborde le problème de la recherche d'information sur Internet et la manière dont il est traité dans la littérature. Ce domaine de recherche est relativement récent au regard des sciences informatiques mais certaines méthodes plus anciennes ayant trait à l'analyse de textes peuvent y être appliquées. Ce chapitre est organisé de manière à présenter les différents aspects du problème : de la formulation d'une requête et de la définition de l'évaluation de documents, en passant par les algorithmes de recherche et de découverte de réponses pertinentes, à la présentation et la visualisation des résultats. Les méthodes et les analyses présentées ici ont servi de base à l'élaboration d'une nouvelle méthode de recherche de documents électroniques dans un vaste réseau fortement connecté.

Depuis son origine, le nombre de documents présents sur Internet n'a cessé d'augmenter. On estime aujourd'hui le nombre de pages accessibles sur ce réseau à quelques dizaines de milliards en 2003. La recherche d'une information particulière parmi cet ensemble est compliquée par la distribution de ces données sur un grand nombre de sites ; on estime qu'il existe environ 50 millions de serveurs sur Internet en avril 2004 [Netcraft]. Il n'est donc plus possible de se passer de l'utilisation d'outils automatiques de recherche d'information. On distinguera dans la suite trois principaux types d'outils de recherche de natures bien différentes tant dans la manière de les interroger que dans leurs méthodes de recherche et de présentation des documents demandés.

Les moteurs de recherche *classiques* permettent à l'utilisateur de formuler une requête de recherche par l'intermédiaire de listes de mots-clés. La recherche consiste à sélectionner dans un corpus - l'index - l'ensemble des documents dans lesquels figurent ces mots et à les trier suivant une mesure définissant leur pertinence à la requête.

Les *sites portails* offrent à l'utilisateur une hiérarchie de documents classés par thèmes. Cette classification peut s'effectuer de manière manuelle - selon l'exemple de *Yahoo !* [Yahoo] - ou automatique - tel l'annuaire *Google* [Google] - suivant une taxonomie prédéfinie. La recherche consiste à se laisser guider dans la hiérarchie de thèmes en sous-thèmes jusqu'à obtenir la réponse désirée. Généralement, un index est associé à la taxonomie de la hiérarchie pour permettre une recherche de thème à base de mots-clés.

Les *méta-moteurs* de recherche offrent à l'utilisateur une interface d'interrogation beaucoup plus élaborée que les moteurs de recherche classiques. La recherche consiste à interroger plusieurs sources de documents - issues des deux premières catégories présentées ci-dessus - en adaptant la requête pour chacun d'eux. Les résultats obtenus sont fusionnés et analysés de manière beaucoup plus précise en fonction de la requête initiale afin de les trier de façon plus pertinente pour l'utilisateur.

Pour chacun de ces trois types d'outils de recherche, plusieurs composantes peuvent être définies. Chacune d'elles représente un domaine de recherche indépendant des autres. Ce chapitre aborde tous les aspects d'un moteur de recherche, de la formulation de la requête à l'affichage des résultats. Il est, de ce fait, relativement conséquent ; nous donnons ainsi au lecteur un petit guide en facilitant la lecture.

- Dans la section 1.1, nous examinons différentes formulations de requêtes utilisables dans un moteur de recherche. Une requête peut être décrite 1) par plusieurs mots-clés associés à des éléments de la logique booléenne, 2) par un document (on recherche alors des documents similaires à celui donné dans la requête), 3) ou par une question formulée dans le langage naturel.
- Dans la section 1.2, nous proposons un état de l'art des différentes techniques et algorithmes de recherche utilisés dans les grands moteurs de recherche actuellement en production. Nous détaillerons dans un premier temps plusieurs études portant sur l'analyse de la topologie du réseau Internet. Nous examinerons ensuite plus particulièrement les algorithmes *HITS*, *PageRank*, *SALSA* et *PHITS* utilisés à grande échelle dans les outils de recherche. Enfin, nous aborderons les principes généraux aux méta-moteurs.
- La section 1.3 aborde l'aspect matériel en analysant les différentes architectures des moteurs de recherche.
- Nous terminerons ce chapitre par la section 1.4 qui présente diverses méthodes d'affichage de résultats permettant à un utilisateur de tirer le meilleur parti de la recherche produite par un outil de recherche. Nous détaillerons les techniques de présentation par liste de réponses et la visualisation de résultats graphiques évolués en deux et trois dimensions.

1.1 Formulation d'une recherche

Dans un système de recherche d'information (SRI) la formulation d'une requête impose une représentation du document et un modèle d'extraction d'informations. Dès 1958, Lunh, un des pionniers de le recherche en SRI, a établi dans [Luhn, 1958] les bases

de l'hypothèse fondamentale des travaux sur l'extraction et la sélection d'informations : *«le contenu textuel d'un document discrimine le type et la valeur des informations qu'il véhicule»*.

La quasi totalité des SRI actuels se basent sur ce principe. L'analyse de la présence de mots dans un corpus de texte permet de déterminer les documents susceptibles de répondre aux souhaits d'un utilisateur du SRI.

Nous allons ainsi examiner dans cette section les différents systèmes permettant de spécifier la présence, l'absence ou encore la proximité de termes dans un document. Nous étudierons dans un premier temps le système booléen et le système booléen étendu à l'origine des premiers moteurs de recherche. Nous détaillerons ensuite d'autres méthodes autorisant la spécification des requêtes sous une forme différente comme le modèle vectoriel qui permet de mesurer le degré de similarité entre deux documents. Nous nous intéresserons également dans la suite de cette section aux systèmes proposant à l'utilisateur de poser des questions dans le langage naturel. Enfin, nous analyserons le comportement des utilisateurs face à ces systèmes de requêtes à l'aide de statistiques issues des principaux moteurs de recherche commerciaux.

1.1.1 Le modèle booléen

Dès l'apparition des premiers SRI, le modèle booléen [Salton et McGill, 1983] s'est imposé grâce à la simplicité et à la rapidité de sa mise en œuvre. L'interface d'interrogation de la plupart des moteurs de recherche est basée sur ce principe et n'est composée que d'une liste de mots-clés. Ces termes peuvent être combinés à des opérateurs logiques $(\wedge, \vee, \neg)$ afin de déterminer au mieux le souhait d'un utilisateur.

Dans ce modèle, un document (d) est représenté par son ensemble de termes (t_i), et une requête (q) comme une expression logique de termes. Un document ne correspondra à une requête que si l'implication $d \Rightarrow q$ est valide. Cette correspondance $C(d, q)$ est déterminée comme suit :

$$\begin{aligned} C(d, t_i) &= 1 \quad \text{si } t_i \in d\,;\, 0 \text{ sinon} \\ C(d, q_1 \wedge q_2) &= 1 \quad \text{si } C(d, q_1) = 1 \text{ et } C(d, q_2) = 1\,;\, 0 \text{ sinon} \\ C(d, q_1 \vee q_2) &= 1 \quad \text{si } C(d, q_1) = 1 \text{ ou } C(d, q_2) = 1\,;\, 0 \text{ sinon} \\ C(d, \neg q) &= 1 \quad \text{si } C(d, q) = 0\,;\, 0 \text{ sinon} \end{aligned} \tag{1.1}$$

En pratique, les termes de la totalité des documents sont identifiés et stockés en conservant les liaisons d'appartenance à chaque texte. On désigne cet ensemble sous le terme *index inversé*. La recherche des documents dans lesquels figure un terme est ainsi fortement accélérée.

Pour permettre à l'utilisateur de spécifier au mieux sa requête, certains moteurs de recherche ont ajouté d'autres opérateurs logiques au modèle initial adaptés plus particulièrement au contexte et à la problématique textuelle. Ce sont les opérateurs de proximité - ou d'adjacence - et les opérateurs de troncature. Ils permettent de préciser la position de deux termes l'un par rapport à l'autre pour les premiers et la recherche

de termes syntaxiquement proches pour les derniers. Ils sont généralement utilisés sous la forme de guillemets (") ou d'astérisques (*) dans les formulations de requêtes.

Une des principales faiblesses du modèle booléen est de distinguer uniquement l'adéquation d'un document à une requête et de ne pas établir de relation d'ordre de pertinence entre les différents éléments du corpus. C'est pour cette raison que ce modèle est généralement utilisé dans la première étape d'un SRI pour pré-sélectionner les documents résultant de la requête initiale afin d'y appliquer un algorithme de tri par pertinence.

1.1.2 Le modèle booléen étendu

Le modèle booléen étendu a été introduit par Salton [Salton *et al.*, 1983]. C'est une extension du modèle précédent qui vise à tenir compte d'une pondération des termes dans le corpus. Cela permet de combler le manque du modèle standard en ordonnant les documents retrouvés par le SRI.

La requête reste inchangée et est composée d'une expression booléenne classique. Par contre, la représentation d'un document se voit augmentée par l'ajout de pondération pour chaque terme (t_i, w_i). En général ce poids est principalement basé sur le nombre d'occurrences d'un terme dans le document mais dans certains systèmes la typographie - taille du texte, forme de la police de caractère - est également prise en compte.

La détermination de la correspondance d'un document à une requête $C(d, q)$ peut prendre plusieurs formes. Suivant le cadre classique des ensembles flous proposé par Zadeh [Zadeh, 1965], on obtient les relations suivantes :

$$\begin{aligned} C(d, t_i) &= a_i \\ C(d, q_1 \wedge q_2) &= min(C(d, q_1), C(d, q_2)) \\ C(d, q_1 \vee q_2) &= max(C(d, q_1), C(d, q_2)) \\ C(d, \neg q) &= 1 - C(d, q) \end{aligned} \tag{1.2}$$

Les opérateurs logiques $\wedge$ et $\vee$ sont évalués respectivement par min et max. Cependant, cette évaluation n'est pas parfaite. On n'obtient pas les relations $C(d, q \wedge \neg q) \equiv 0$ et $C(d, q \vee \neg q) \equiv 1$. Ce qui signifie que lorsqu'on évalue une requête sous forme de conjonction, on ne s'intéresse qu'à la partie la plus difficile et lorsqu'on évalue une requête sous forme de disjonction, c'est la partie la plus facile qui domine. C'est pour cette raison que d'autres formes de correspondances ont été proposées. Les relations les plus utilisées ont été introduites par Lukasiewicz [Lukasiewicz, 1963] sous la forme suivante :

$$\begin{aligned} C(d, t_i) &= a_i \\ C(d, q_1 \wedge q_2) &= C(d, q_1) \cdot C(d, q_2) \\ C(d, q_1 \vee q_2) &= C(d, q_1) + C(d, q_2) - C(d, q_1) \cdot C(d, q_2) \\ C(d, \neg q) &= 1 - C(d, q) \end{aligned} \tag{1.3}$$

Les deux parties d'une conjonction ou d'une disjonction contribuent en même temps à l'évaluation de la correspondance du document à la requête. Cependant, ce modèle n'est pas parfait (on n'obtient toujours pas $C(d, q \wedge \neg q) \equiv 0$ et $C(d, q \vee \neg q) \equiv 1$) mais il reste convenable.

Le modèle p-norme introduit par Salton mesure les correspondances de la conjonction et de la disjonction [Salton *et al.*, 1983]. L'idée de base réside dans l'observation de la table de vérité (voir table 1.1). Dans la colonne $A \wedge B$ la meilleure correspondance est atteinte dans le cas de la dernière ligne. Dans la colonne $A \vee B$ la pire correspondance correspond à la première ligne.

A	B	A ∧ B	A ∨ B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

TAB. 1.1 – Table de vérité.

On peut ainsi considérer que l'évaluation d'une conjonction ou d'une disjonction consiste à calculer une sorte de distance entre le point à atteindre ou à éviter. Ce concept est illustré par la figure 1.1. Dans ces schémas, les axes des abscisses correspondent à l'évaluation du document A et les axes des ordonnées à l'évaluation du document B.

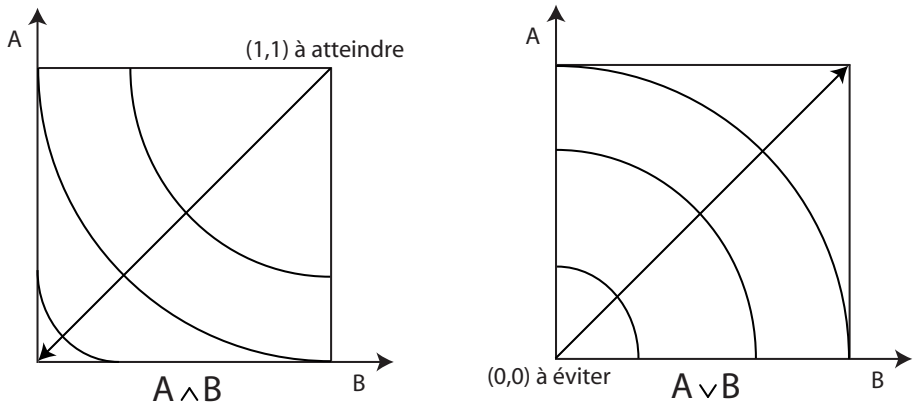


FIG. 1.1 – Évaluation d'une conjonction ou d'une disjonction.

Pour la conjonction, on cherche à évaluer dans quelle mesure le point c défini par l'évaluation d'un document A et d'un document B est proche de (1,1), le point à atteindre. Ce rapprochement peut être mesuré par le complément de la distance entre le point c et le point (1,1). Plus cette distance est grande, moins A ∧ B est satisfaite. Les points qui se situent sur une même courbe, ont la même distance avec (1,1) et ainsi correspondent à la même évaluation. Dans le cas de A ∨ B, on recherche plutôt à éviter le point (0,0). Plus on est loin de (0,0), plus A ∨ B est satisfaite. Ainsi Salton propose

les relations de correspondance normalisées suivantes :

$$\begin{aligned} C(d,t_i) &= a_i \\ C(d,q_1 \wedge q_2) &= 1 - \sqrt{\frac{(1-C(d,q_1))^2 + (1-C(d,q_2))^2}{2}} \\ C(d,q_1 \vee q_2) &= \sqrt{\frac{C(d,q_1)^2 + C(d,q_2)^2}{2}} \\ C(d,\neg q) &= 1 - C(d,q) \end{aligned} \tag{1.4}$$

Le principal avantage de ce modèle sur le modèle booléen classique réside dans la mesure du degré de correspondance entre un document et une requête dans $[0,1]$. On peut ainsi ordonner les documents dans l'ordre décroissant de leur correspondance avec la requête.

Ce modèle de tri n'est pratiquement plus implémenté en tant que tel dans les moteurs de recherche actuels. L'ordre établi est maintenant remplacé par le tri de pertinence associé à la topologie des liens hypertextes dans un réseau. Nous examinerons cette technique dans la section 1.2.

1.1.3 Le modèle vectoriel

L'interface d'interrogation d'un moteur de recherche par des listes de mots-clés montre ses limites dès lors qu'un utilisateur n'assimile pas le système booléen et n'a pas une idée précise des mots les plus pertinents caractérisant son souhait. Certains méta-moteurs de recherche ont commencé à explorer une autre voie en proposant une nouvelle interface d'interrogation permettant à l'utilisateur de spécifier des pages ressemblant aux résultats qu'il attend. Par exemple, le système *Webnaut* [Nick et Themis, 2001] scrute les pages visitées par un utilisateur et les utilise pour guider ses recherches. Nous décrirons ce méta-moteur avec plus de détails dans la section 2.4.1.1 du chapitre 2. Cette démarche à l'avantage de cerner d'une manière beaucoup plus précise les souhaits d'un utilisateur. Cependant, elle nécessite l'établissement de méthodes permettant de déterminer la similarité entre deux documents.

La notion de similarité entre documents est fortement liée au choix de la méthode de représentation des textes. Dans le cas de collections de textes de grande taille, la représentation admettant actuellement les meilleurs résultats et la plus utilisée est la représentation vectorielle. Elle a été mise en œuvre dès 1971 par Salton dans le système de recherche documentaire SMART [Salton, 1971, Rocchio, 1971]. Dans cette méthode, un document est représenté par un vecteur dans un espace vectoriel dont les dimensions sont associées à des unités linguistiques spécifiques (mot, lemmes, etc.) que l'on désigne dans la littérature par *terme d'indexation*. Ce choix de terme d'indexation n'est pas aisé. Il représente un facteur prépondérant dans la détermination de l'efficacité de la mesure de similarité entre documents. Il repose principalement sur des systèmes d'analyse de texte et de détermination du sens lexical des documents.

Les premiers travaux conséquents dans ce domaine ont été proposés par Luhn. Dans un de ses premiers papiers [Luhn, 1958], il a énoncé que la fréquence d'apparition d'un

mot dans un document établit une mesure efficace de la signification des documents. Il a également proposé que la position relative de mots apportant le plus d'information dans une phrase fournit une bonne mesure pour déterminer la signification des phrases. La détermination du facteur de signification d'un document est ainsi basée sur ces deux mesures.

Dans le domaine de l'analyse automatique de textes, Luhn a donc contribué à établir que la fréquence des termes d'un document peut être utilisée pour extraire les mots ou phrases qui représentent ce document.

L'analyse statistique de nombreux textes a montré que la fréquence d'apparition d'un mot dans un corpus en fonction de son rang dans le document - fonction décroissante du nombre d'occurrences du mot - forme une courbe similaire à une hyperbole. La courbe de fréquence de la figure 1.2 illustre cette situation. Cette courbe suit en réalité la loi de Zipf [Zipf, 1949] qui établit que le produit de la fréquence d'un terme par son rang est approximativement constant. Cette loi a été vérifiée par Zipf sur la base de journaux américains en anglais mais s'applique sur tout corpus linguistique.

Or dans un texte, la valeur informative d'un mot peut s'exprimer sous la forme d'une gaussienne en fonction du rang des termes d'un document comme montré dans le figure 1.2 [Van Rijsbergen, 1979]. En effet, les mots les plus fréquents d'un texte représentent généralement des mots de liaison ou d'usage courant et ne véhiculent pas ou peu d'information sur la signification d'un document, et les mots les moins fréquents n'apportent que rarement un sens primordial à la compréhension d'un texte.

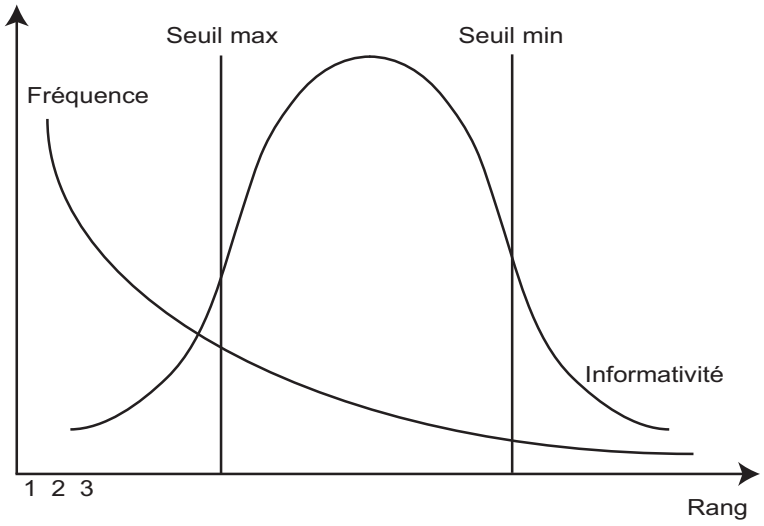


FIG. 1.2 – Relation entre le rang et la fréquence d'apparition d'un terme dans un document.

Luhn a utilisé cette constatation comme hypothèse de base pour spécifier deux seuils de coupure (seuil max et seuil min sur la figure) déterminant les mots apportant le plus d'information pour caractériser le sens d'un document et éliminant les mots sans

signification. Les mots au-delà du seuil maximum sont considérés comme trop communs et ceux en deçà du seuil minimum comme trop rares. La probabilité d'obtenir un mot discriminant le contenu d'un document est maximale entre les deux seuils et tend à diminuer dans toutes les directions atteignant des valeurs proches de zéro aux points de coupure. Il n'y a pas de solution idéale pour établir ces seuils et la meilleure méthode consiste à procéder par essais successifs par rapport à un corpus donné. Cependant, dans le cas général, l'intervalle $\left[\frac{|\mathcal{C}|}{100}, \frac{|\mathcal{C}|}{10}\right]$ où $|\mathcal{C}|$ est le nombre de mots dans le corpus, est considéré comme adéquat pour fournir des termes avec un bon pouvoir discriminant [Salton *et al.*, 1975].

Ces idées simples sont à la base de la plus grande partie des travaux réalisés dans le domaine de la recherche d'information. Luhn les a lui-même utilisées pour construire une méthode de génération automatique de résumés [Luhn, 1958]. L'algorithme consiste à déterminer la proportion de mots significatifs ou non dans chaque portion de texte puis à les trier en fonction de leur score. Seules les meilleures phrases sont alors insérées dans le résumé.

Dans le cadre du modèle vectoriel, chaque document d est ainsi représenté au moyen d'un vecteur $\langle d \rangle = \left(\omega_{t_1}, \ldots, \omega_{t_{|\mathcal{T}|}}\right)$ dénommé dans la littérature *profil lexical*, où ω_{t_i} représente le poids, dans le document d, du terme d'indexation t_i associé à la $i^{ème}$ dimension de l'espace vectoriel.

Le poids d'un profil lexical t_i peut être représenté par plusieurs mesures. La plus connue et utilisée à ce jour est certainement la mesure *tf*idf* [Salton et Buckley, 1988]. Elle regroupe un ensemble de schémas de pondération et de sélection de termes. *tf* signifie 'term frequency' et *idf* 'inverted document frequency'.

On retrouve dans la littérature plusieurs formules *tf* et *idf*. Certaines introduisent une normalisation, mais celle-ci n'est pas nécessaire dans le cas de l'utilisation d'une mesure de similarité indépendante de la norme. Généralement on utilise les formules suivantes pour calculer les éléments *tf*idf* d'un document :

$$tf = log\left(f(t_i, d) + 1\right) \tag{1.5}$$

$$idf = log(1/n) \tag{1.6}$$

Avec $f(t_i, d)$ correspondant à la fréquence d'occurrence du terme t_i dans le document d et n représentant le nombre de documents du corpus dans lequel figure le terme t_i.

Une formule *tf*idf* combine les deux critères précédents. La composante tf indique l'importance du terme pour un document tandis que *idf* indique le pouvoir de discrimination de ce terme. Ainsi, un mot ayant une valeur de *tf*idf* élevée doit être à la fois important dans ce document et apparaître peu dans les autres documents.

$$\omega_{t_i} = tf * idf = log\left(f(t_i, d) + 1\right) \cdot log(1/n) \tag{1.7}$$

Cette représentation des documents sous forme de vecteurs de termes d'indexation nous permet de déterminer la similarité entre les différents éléments de notre corpus. La

mesure *cosine* [Salton et McGill, 1983] est utilisée dans la plupart des travaux recensés dans ce domaine. Cette méthode consiste à calculer le cosinus de l'angle entre le vecteur représentant la requête de l'utilisateur et chaque document du corpus. Elle se caractérise par la formule suivante :

$$cos(d_i, d_j) = \frac{\sum_n d_{i,n} d_{j,n}}{\sqrt{\sum_n d_{i,n}^2 \sum_n d_{j,n}^2}} \tag{1.8}$$

D'autres mesures de similarité peuvent être utilisées, citons par exemple les méthodes basées sur le nombre de concepts communs entre la requête et le document, l'utilisation de la somme des produits *tf*idf* ou encore la mesure Okapi [Robertson *et al.*, 1992]. Mais c'est la mesure *cosine* qui apporte dans la plus grande majorité des cas les meilleurs résultats [Bellot, 2000].

Nous verrons par la suite des exemples de moteurs de recherche utilisant ces mesures de similarités afin de proposer à l'utilisateur de formuler sa requête à l'aide de documents ressemblant à ceux qu'il désire retrouver.

1.1.4 Requêtes à base de questions

Le langage booléen est parfaitement adapté au traitement informatique de données. Cependant, il n'offre pas l'étendue des possibilités d'expression du langage naturel. C'est pourquoi des recherches sont poursuivies dans le domaine de la reconnaissance du sens d'une phrase posée en langage naturel et de sa traduction dans une syntaxe plus compréhensible par un outil de traitement automatique.

Une première piste consiste à étendre le langage booléen à un autre langage plus développé. Un des plus connu et utilisé est certainement *WebSQL*. Ce langage a été introduit par Mihaila [Mihaila, 1996]. Il se base sur un formalisme similaire à celui développé dans le langage SQL d'interrogation de base de données.

WebSQL considère ainsi le Web comme une base de donnée relationnelle composée de deux tables : des documents et des liens hypertextes. La table des documents est composée d'un *t-uple* pour chaque document présent sur Internet - un *t-uple* comprend une url, un texte, un type de document, ... - et la table des liens hypertextes d'un *t-uple* pour chaque lien de chaque document du réseau. Les requêtes peuvent alors être formulées sur cette base de données virtuelle d'une manière similaire à la syntaxe employée dans le langage SQL. La figure 1.3 montre l'exemple d'une requête réalisée à l'aide de ce langage demandant deux documents d'URL différentes dont les titres sont les mêmes et dans lesquels figurent les mots «something interesting».

Ce langage a été implémenté dans une application réelle afin de démontrer la faisabilité d'un tel procédé [Arocena *et al.*, 1997].

Une autre piste, plus difficile, consiste à analyser une question posée directement en langage naturel. Par exemple, le moteur de recherche doit être capable de répondre à la question : *«Quels sont les films à l'affiche ce soir ?»*.

```
SELECT d1.url, d2.url
FROM   Document d1 SUCH THAT d1 MENTIONS "something interesting",
       Document d2 SUCH THAT d2 MENTIONS "something interesting"
WHERE d.title = d2.title
AND NOT (d1.url = d2.url)
```

FIG. 1.3 – Exemple de requête formulée dans le langage *WebSQL*.

Kwok a proposé dans [Kwok *et al.*, 2001] un système se basant sur ce principe qu'il a nommé *Mulder*. Pour arriver à ses fins, *Mulder* traite tout d'abord la requête - en langage naturel - et construit un arbre en fonction de la structure de la phrase. Ce système est dépendant des règles de formation des phrases spécifiques à une langue. Cet arbre est ensuite analysé par un classifieur qui détermine le type de réponse attendue : nominale, numérique ou temporelle demandant respectivement une réponse sous la forme d'une phrase, d'un nombre ou d'une date.

L'étape suivante consiste à formuler une ou plusieurs requêtes dans un langage compréhensible par les moteurs de recherche actuels. Ces requêtes sont composées de termes devant être contenus dans la réponse. En effet, un moteur de recherche classique est un indexeur. Lorsque l'on veut connaître la réponse à une question, il faut chercher la réponse dans une base de données et non pas la question relative à cette réponse. Ainsi, pour l'exemple donné dans le paragraphe précédent, on devrait idéalement reformuler la requête d'interrogation en *«Les films à l'affiche ce soir sont»*. Pour réaliser cela, le système *Mulder* reformule la question en utilisant des règles spécifiques à la langue comme un système d'extraction et de conjugaison de verbes.

Le système crée de cette manière plusieurs requêtes allant d'une forte spécialisation, utilisant beaucoup de termes de la question initiale, à une requête générale, contenant très peu de termes. Ces différentes requêtes permettent de s'affranchir du domaine traité et d'obtenir des réponses sans que celles-ci ne soient trop vagues - caractérisées par de très nombreuses réponses. En effet, si la spécialisation est importante et donc si le nombre de termes de la requête est trop important, il est possible qu'un moteur de recherche ne puisse fournir le moindre résultat.

Une fois les résultats des moteurs de recherche obtenus, *Mulder* produit un résumé en sélectionnant les zones des documents retournés dans lesquelles figurent les termes de la requête initiale. Ce système permet de laisser l'utilisateur apprécier la justesse de la réponse.

Mulder a montré son efficacité en obtenant des résultats en moyenne six fois plus rapidement que les moteurs de recherche classiques. Mais il est nécessaire de produire des règles de transformation de question pour chaque langue prise en compte, ce qui représente un travail relativement fastidieux.

Chaque moteur de recherche emploie son propre système de requête, du plus simple au plus complexe. Cependant, l'utilisateur final, n'est pas forcément apte à utiliser n'importe quel système. En effet, un moteur de recherche classique ne s'adresse pas à

un public de spécialistes mais à l'ensemble des utilisateurs d'Internet, du plus novice au plus expert. Des études statistiques ont été réalisées sur l'ensemble des requêtes formulées sur des moteurs de recherche utilisés par des millions d'utilisateurs à travers le monde afin de vérifier les aptitudes des utilisateurs et leur comportement vis-à-vis de la manière dont leur sont fournis les résultats. La section suivante présente ces résultats sur deux moteurs de recherche très utilisés : *Altavista* et *Excite*.

1.1.5 Statistiques d'utilisation

Plusieurs études ont cherché à analyser le comportement des utilisateurs face à ces systèmes de requêtes. Les deux plus grosses analyses ont été réalisées sur les moteurs de recherche *Excite* [Spink *et al.*, 2001] et *Altavista* [Silverstein *et al.*, 1998].

Ces deux études, bien que ne donnant pas exactement les mêmes chiffres, dégagent des tendances similaires. Par conséquent, les résultats exposés dans la suite de cette section concernent uniquement l'étude menée sur le moteur *Excite*, étant plus récente et portant sur un plus gros volume de données que celle proposée sur *Altavista*. Elle s'intéresse tout particulièrement à un ensemble de 1 025 910 requêtes proposées par 211 063 utilisateurs du système. L'intérêt de cette étude est de savoir ce que cherchent les utilisateurs et comment ils procèdent pour effectuer cette recherche. Pour ce faire, les collections de transactions de recherche sur le moteur sont analysées afin de déterminer le comportement des utilisateurs.

Parmi les requêtes analysées, 51,8% sont uniques, 38,5% sont répétées et 9,7% ne contiennent aucun terme de recherche. Statistiquement, les utilisateurs proposent 4,86 requêtes dans une session de recherche. Cela signifie qu'il leur faut effectuer près de 5 requêtes en moyenne sur le moteur de recherche pour être satisfaits du résultat proposé. Cependant, ce résultat peut être nuancé par le fait que 48,4% des utilisateurs ne soumettent qu'une seule requête, 20.8% deux requêtes et 31% plus de deux requêtes.

Il est intéressant de regarder de quelle manière les requêtes sont modifiées parmi les 52% d'utilisateurs qui effectuent plus d'une requête par session. Dans 32,5% des requêtes modifiées, le nombre total de termes ne change pas, dans 41,6% des termes ont été rajoutés à la requête initiale et dans 25,9% des requêtes des termes ont étés supprimés. Mais en général, peu de termes sont ajoutés ou supprimés : 99,2% des utilisateurs n'ajoutent ni ne suppriment plus de 5 termes. Les modifications effectuées sont toutefois très minimes.

Une autre statistique remarquable concerne le nombre de résultats visionnés par l'utilisateur à l'issue d'une requête. Globalement peu de pages sont examinées : 28,6% des utilisateurs n'examinent qu'une seule page de résultats alors que 10 liens par page de résultats sont renvoyés, 19% examinent deux documents. Quasiment la moitié des utilisateurs n'examinent que deux liens parmi tous les résultats renvoyés. De plus, une écrasante majorité des utilisateurs n'examinent pas plus d'une page de résultats. Les statistiques menées sur le moteur *Altavista* montrent même que près de 96% des utilisateurs n'examinent pas plus de trois liens.

Ces statistiques ne signifient pas forcément que les résultats des recherches sont sa-

tisfaisants pour l'utilisateur. En effet, devant un très grand nombre de résultats fournis, ce dernier peut également considérer que la précision du moteur n'est pas correcte pour sa demande. De plus les moteurs de recherche affichent un résumé de quelques lignes permettant de se faire une idée de la pertinence du lien proposé sans forcément aller examiner le document lui-même.

Cette étude montre également que seulement 4% des utilisateurs effectuent des requêtes en utilisant le formalisme de la logique booléenne. De même, 75% des requêtes ne contiennent rien d'autre que des termes de recherche. Aucun autre opérateur de recherche n'est utilisé dans ces requêtes. Peu d'utilisateurs acceptent d'apprendre le fonctionnement du système de requête d'un moteur pour améliorer la formulation de leurs requêtes.

Excite propose un système capable de retourner des pages similaires à un lien donné en résultat. Mais seulement 5% des utilisateurs effectuent cette démarche pour un total de 9,7% des requêtes analysées. Il s'agit là d'une faible utilisation de cette possibilité et signifie à la fois que les utilisateurs ne retrouvent pas de sites très pertinents, ne prêtent pas attention à la recherche de documents similaires à un bon résultat ou ne comprennent pas l'utilité d'une telle fonction.

L'analyse de l'ensemble des requêtes proposées par les utilisateurs montre que 57,1% des termes utilisés ne se rencontrent qu'une seule fois, 14,5% deux fois et 6,7% trois fois. Ainsi 78,3% des 140 279 termes uniques détectés sont utilisés moins de quatre fois sur plus d'un million de requêtes analysées. Les 75 termes les plus fréquents ne représentent que 0,05% des termes utilisés mais apparaissent dans 9% des mots employés dans les requêtes proposées au moteur de recherche. Les 15 termes les plus employés sont assez explicites : *«sex, nude, free, pictures, new, university, women, chat, gay, girls, xxx, music, software, pics, ncaa»* et se retrouvent sur la quasi totalité des analyses effectuées sur d'autres moteurs de recherche.

Ainsi, toutes les études montrent que l'usage le plus courant d'un moteur de recherche ne porte que sur un nombre limité de termes et donc de sujets. Mais d'un autre côté, le vocabulaire utilisé dans l'ensemble des requêtes est très étendu et couvre très largement celui utilisé dans la majorité des textes présents sur le réseau.

Quel que soit le type de requête proposé, un algorithme de recherche tente de déterminer les documents les plus pertinents répondant à la question posée. Dans la section suivante, nous allons étudier les différentes techniques utilisées à l'heure actuelle et leur implémentation dans des systèmes opérationnels. Ces systèmes se basent sur différentes notions pour établir le degré de pertinence d'un document, allant de l'analyse linguistique et statistique des termes contenus dans le document à la nature des liens existants entre plusieurs documents.

1.2 Algorithmes de recherche pour Internet

Les moteurs de recherche sur Internet doivent faire face à de très fortes contraintes. Les outils de recherche doivent répondre à des millions de requêtes par jour alors que la quantité de documents qu'ils doivent analyser est gigantesque (à cette date *Google* [Google], le moteur de recherche le plus populaire, indexe près de 4,3 milliards de pages Web). La détection d'une mise à jour de ces documents est elle aussi source de grandes difficultés. Il n'y a pas de centralisation de ces données et un moteur de recherche doit par conséquent scruter en permanence le réseau en vue de détecter ces changements. Cette problématique est bien illustrée par une citation de Sergey Brin, un des inventeurs de *Google* et de l'algorithme *PageRank* [Brin et Page, 1998] : *«Le Web est une vaste collection de documents hétérogènes complètement incontrôlés»*.

Pour toutes ces raisons, des recherches ont été menées afin d'alléger la charge des machines composant les systèmes de recherche. Le principal facteur de ralentissement étant l'analyse des textes afin d'établir un tri de pertinence de résultats, d'autres voies ont été explorées. Les meilleurs résultats ont été obtenus par des méthodes utilisant les propriétés déduites de l'analyse de la topologie d'Internet.

Dans cette section, nous examinerons dans un premier temps les variables caractérisant Internet et les documents qui composent ce réseau. Ensuite nous étudierons plusieurs algorithmes de détection de pertinence employés dans les moteurs de recherche. Enfin nous évoquerons le cas des méta-moteurs et leurs différences avec les moteurs de recherche classiques.

1.2.1 Topologie d'Internet

De nombreuses études ont été réalisées à la fin des années 1990 dans le but de déterminer et de caractériser les connexions établies par les liens hypertextes des documents d'Internet. Ce réseau diffère fondamentalement des autres réseaux connus de par sa nature ouverte et son utilisation grandissante.

Le développement d'Internet et plus particulièrement du *World-Wide Web* est complètement incontrôlé. Chaque individu peut créer son propre site Web avec le nombre de documents et de liens hypertextes qu'il souhaite. Cette situation favorise une augmentation continue du nombre de pages et de liens disponibles et mène à la création d'un vaste réseau très complexe.

Il existe principalement deux manières de modéliser la topologie d'Internet sous forme d'un graphe. La première consiste à considérer les nœuds de routage du réseau comme les nœuds du graphe et les liaisons entre les routeurs comme des arcs non orientés. Cette modélisation a un intérêt dans l'optimisation des algorithmes de routage d'un réseau [Calvert *et al.*, 1997] mais peu dans l'optique de la recherche d'information.

La seconde s'intéresse non pas aux connexions physiques du réseau mais aux relations qui existent entre les documents d'Internet. Cette modélisation représente Internet par un graphe orienté dont les nœuds et les liens sont respectivement représentés par les

documents - contenu textuel, structure ... - et les URL des liens hypertextes qui relient un document à un autre. La topologie de ce graphe détermine la connectivité d'Internet et par conséquent nous renseigne sur la meilleure manière de rechercher de l'information dans ce réseau. Cependant, sa taille gigantesque, estimée à plusieurs milliards de nœuds et sa perpétuelle évolution tant au niveau des arcs que des nœuds rend impossible l'analyse de l'ensemble de ces éléments.

Par conséquent la plupart des travaux réalisés dans ce domaine ont tenté de déduire les lois régissant ce réseau à partir d'un sous-graphe d'Internet. Ces lois sont ensuite vérifiées et validées par des simulations de réseaux à plus grande échelle afin de déduire les propriétés et les caractéristiques du graphe complet d'Internet.

Une des premières constatations réalisée par l'étude de ce graphe est que plusieurs lois de puissance régissent Internet. Une loi de puissance est une expression de la forme $y \propto x^a$, où a est une constante, x et y sont des mesures effectuées sur le sujet et $\propto$ signifie *proportionnel à*.

Dans [Faloutsos *et al.*, 1999, Siganos *et al.*, 2003], Faloutsos et Siganos ont déterminé trois lois de puissance caractérisant Internet au niveau des connexions existant entre les serveurs de données. Leur raisonnement a montré qu'il existait deux niveaux régissant Internet selon des paramètres différents. On distingue ainsi un haut niveau où plusieurs nœuds de communication s'entre-connectent et un plus bas niveau, à l'intérieur de chaque nœud, où plusieurs serveurs de données sont fortement reliés entre eux. La figure 1.4 illustre cette situation. Les expérimentations ont été réalisées sur plusieurs sous-graphes d'Internet répartis en fonction des sous-domaines constatés et du graphe global.

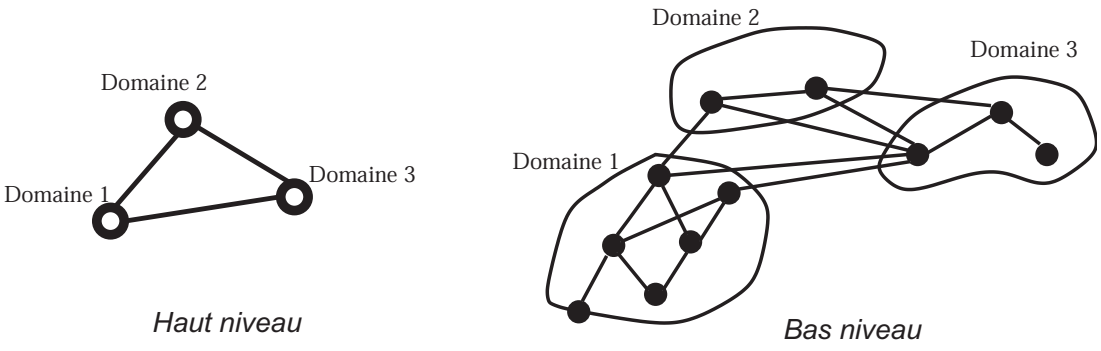


FIG. 1.4 – Structure d'Internet à deux niveaux.

Dans le graphe orienté représentant Internet, chaque nœud possède des liens *sortants* et des liens *entrants*. La première loi de puissance cherche à caractériser l'organisation de ces liens. Pour cela, les nœuds sont triés dans l'ordre décroissant de leur nombre de liens sortants afin de leur attribuer un rang. Cette loi peut alors s'exprimer ainsi : *Le nombre de liens sortants d_v d'un nœud v est proportionnel au rang de ce nœud r_v à la puissance d'une constante $\mathcal{R}$* (équation 1.9).

$$d_v \propto r_v^{\mathcal{R}} \tag{1.9}$$

Expérimentalement, deux valeurs de la constante $\mathcal{R}$ ont été établies. Elles caractérisent les deux niveaux de connexion d'Internet : $\mathcal{R} \approx -.80$ si on ne considère que les inter-domaines - liaisons internes à un domaine en particulier - et $\mathcal{R} \approx -.48$ si on considère le graphe complet des liaisons.

Cette loi reflète la manière dont les domaines d'Internet sont connectés. Elle traduit l'équilibre entre le gain et le coût de l'ajout d'un arc d'un point de vue financier et fonctionnel. Grâce à elle on peut estimer, connaissant le nombre total de nœuds N, le nombre d'arcs E en fonction de la constante $\mathcal{R}$ (équation 1.10).

$$E = \frac{1}{2(\mathcal{R}+1)} \left(1 - \frac{1}{N^{\mathcal{R}+1}}\right) N \tag{1.10}$$

La deuxième loi de puissance établit une relation décrivant le processus d'ajout d'un arc à un nœud du graphe. Pour cela on considère la fréquence f_d d'un nombre de liens sortant d - le nombre de nœuds possédant d liens sortants - en fonction du nombre de liens sortants d. Cette loi s'exprime par l'équation 1.11 où $\mathcal{O}$ est une constante établie expérimentalement. Elle traduit le fait que les nœuds possédant peu de liens sortants sont plus nombreux que les autres.

$$f_d \propto d^{\mathcal{O}} \tag{1.11}$$

Intuitivement, cette loi se justifie en constatant que la distribution des liens sortants sur Internet n'est pas arbitraire. Expérimentalement, on retrouve également deux valeurs de la constante : $\mathcal{O} \approx -2.18$ et $\mathcal{O} \approx -2.48$ respectivement pour le graphe des inter-domaines et le graphe complet d'Internet.

La troisième loi s'intéresse aux valeurs propres des graphes d'Internet pris en considération. Cette caractérisation est importante car il est prouvé dans la littérature que les valeurs propres traduisent plusieurs caractéristiques et propriétés topologiques d'un graphe comme son diamètre, le nombre d'arcs, l'arbre de couverture, le nombre de composants connectés et le nombre de chemins d'une certaine taille entre les nœuds du graphe [Cvetkovic̀ *et al.*, 1979].

L'observation des valeurs propres λ_i en fonction de leur rang i a permis d'établir l'équation 1.12 où $\mathcal{E}$ est une constante définie expérimentalement :

$$\lambda_i \propto i^{\mathcal{E}} \tag{1.12}$$

Là encore, deux valeurs distinctes de $\mathcal{E}$ ont été établies en fonction du graphe pris en compte : $\mathcal{E} \approx -0.48$ pour chacun des sous-graphes des domaines d'Internet et $\mathcal{E} \approx -0.18$ pour le graphe complet. Ces valeurs établies expérimentalement confirment qu'il existe bien deux niveaux de connectivité dans le graphe représentant Internet. On décompose ainsi ce réseau en sous réseaux connectés entre eux. Intuitivement, cela représente bien l'historique de construction d'Internet qui a, à son origine, servi à relier plusieurs réseaux entre eux. L'esprit de ce développement existe encore et est similaire au niveau

de la construction des réseaux de documents que l'on peut séparer en plusieurs sites - regroupant chacun de nombreuses pages Web fortement connectées entre elles - reliés par des liens.

Cette vision d'un graphe à plusieurs niveaux a amené certains auteurs à considérer Internet du point de vue des fractales [Yook *et al.*, 2002]. Ainsi, au plus bas niveau, Internet est un réseau de routeurs connectés par des liens. Chaque routeur dépend d'une autorité d'administration ou de systèmes autonomes. Le développement des routeurs est fortement corrélé à la densité de population à travers le monde. Ce n'est pas une coïncidence puisqu'une forte densité de population implique une forte demande de services Internet, résultant en une forte densité de domaines Internet.

Yook a également montré dans son étude que le développement du réseau lui-même suit les propriétés des fractales. Autrement dit, plus un nœud du graphe représentatif d'Internet possède d'arcs entrants, plus de nouveaux nœuds auront tendance à se lier à lui. Cette notion est très importante car elle est à la base de la plupart des algorithmes de détermination de pertinence d'une page dans les moteurs de recherche modernes. En effet, on peut considérer que si un nœud du graphe est pointé par beaucoup de liens, celui-ci possède une caractéristique intéressante. Il agit comme une sorte d'autorité au regard d'autres nœuds plus faiblement connectés. Ainsi, lors de la création d'un nouveau point dans le graphe, la connexion se fera préférentiellement vers une autorité du domaine et donc vers un nœud fortement relié dans le graphe afin d'obtenir une certaine légitimité.

L'examen de la topologie de connexion des pages Web entre elles amène des informations très importantes sur la manière de rechercher de l'information dans les documents présents sur Internet. La vaste distribution des pages Web à travers le monde entier oriente vers une première problématique : comment les documents sont-ils reliés entre eux et comment les atteindre tous ? Le diamètre de notre graphe est ainsi une variable prépondérante du problème. Il permet de connaître le nombre de liens nécessaires à suivre pour atteindre n'importe quel point du graphe depuis un nœud donné. Albert, Jeong et Barabási se sont attachés à ce problème et ont levé le voile sur plusieurs de ces interrogations dans [Albert *et al.*, 1999, Barabási *et al.*, 2000].

Pour déterminer la connectivité des pages du Web, un robot parcourt le graphe du Web en stockant dans une base de données toutes les URLs - les liens - rencontrées dans chaque document. Dans leur étude, les auteurs ont ainsi construit un sous-graphe d'Internet comportant près de 326 000 documents reliés par environ 1 470 000 liens. Les mesures effectuées sur cet ensemble de données ont établi que les probabilités $P_{in}(k)$ et $P_{out}(k)$ qu'un document ait respectivement k liens entrants et k liens sortants, suivent une loi de puissance de plusieurs niveaux de grandeur (équations 1.13 et 1.14). Ceci nous amène à faire un parallèle avec la topologie observée au plan des connexions des routeurs et des serveurs d'Internet. En effet, on retrouve au niveau des documents la même organisation en plusieurs niveaux établie précédemment.

$$P_{in}(k) \propto k^{-\gamma_{in}} \quad (1.13)$$
$$P_{out}(k) \propto k^{-\gamma_{out}} \quad (1.14)$$

Ainsi, plus une page sera pointée par d'autres pages, plus celle-ci aura tendance à être pointée par de nouvelles pages. Intuitivement, une page Web nouvellement créée inclura plus facilement des liens vers des pages Web connues - et donc populaires - avec une très forte connectivité. Ces lois de puissance indiquent également que la probabilité de trouver un document à l'aide d'un grand nombre de liens est significative tant que la connectivité des pages du réseau est relativement importante. Et d'une manière symétrique, la probabilité de trouver des pages populaires - ou pointées par un grand nombre de documents - est non négligeable.

La compréhension des lois régissant la connectivité - et donc la topologie - de ce graphe a permis d'établir une mesure du plus court chemin reliant deux documents au sein d'Internet. Cette mesure caractérise ce que l'on désigne par *diamètre du Web* et définit le nombre moyen de liens qu'il est nécessaire de suivre pour naviguer d'un document à un autre. L'équation 1.15, établie expérimentalement, indique le *diamètre du Web* obtenu en fonction du nombre total de nœuds dans le graphe du réseau.

$$\langle l \rangle = 0.35 + 2.06 log(N) \quad (1.15)$$

Cela montre qu'Internet n'est en réalité qu'un petit monde extrêmement interconnecté, caractéristique d'un système social ou biologique. En considérant qu'Internet comporte environ 5 milliards de pages Web indexées, $\langle l \rangle = 20.33$. On peut ainsi naviguer d'un point de notre graphe de 5 milliards de nœuds à n'importe quel autre point du graphe en suivant 21 liens seulement en moyenne.

Cette faible valeur de $\langle l \rangle$ nous permet de dire qu'un agent de recherche intelligent qui pourrait interpréter efficacement la signification des liens et qui ne suivrait que ceux considérés comme intéressants pourrait trouver n'importe quelle information rapidement en naviguant simplement sur le Web.

On retrouve dans la littérature des articles plus récents qui ont précisé ces résultats [Kleinberg *et al.*, 1999, Kumar *et al.*, 1999, Broder *et al.*, 2000, Kumar *et al.*, 2000]. La plus grande étude topologique d'Internet réalisée à ce jour comporte environ 200 millions de pages et 1,5 milliards de liens. Elle a été effectuée à partir de plusieurs *crawls* du moteur de recherche *Altavista* entre mai et octobre 1999 [Broder *et al.*, 2000].

La première conclusion importante de ces travaux a conforté l'hypothèse que la distribution des liens entrants suit une loi de puissance de la forme $P_{in}(k) \propto k^{-\gamma_{in}}$. La valeur de γ_{in} a été vérifiée sur plusieurs échelles du graphe du Web et est restée constante pour $\gamma_{in} = 2.1$. D'un autre côté, le nombre moyen de liens sortants observé dans les *crawls* [Kleinberg *et al.*, 1999, Kumar *et al.*, 2000] est de 7,2.

L'analyse des 200 millions de nœuds du graphe a également montré qu'environ 90% des pages forment un noyau fortement connecté si on ne tient pas compte de la nature

orientée des liens. Il existe ainsi environ 17 millions de pages qui sont complètement dissociées de la structure fortement connectée que forme Internet. Une analyse plus précise a dénoté quatre familles de pages aux propriétés distinctes illustrées sur la figure 1.5.

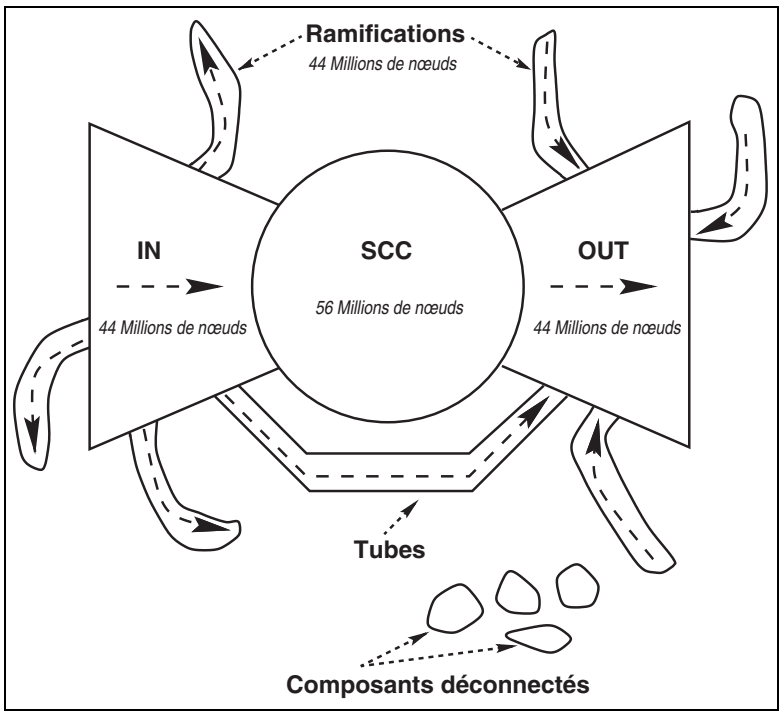


FIG. 1.5 – Représentation du graphe de connexion des pages du Web. 3 groupes se détachent : un noyau fortement interconnecté (SCC), un groupe de pages pointant vers le noyau (IN) et un groupe de pages pointées par le noyau (OUT).

Une première zone centrale dénommée *SCC* se détache du reste du graphe. Il s'agit de pages fortement connectées entre elles. Chacune peut atteindre une autre page en suivant quelques liens hypertextes. On peut aisément considérer ce noyau comme le cœur du Web. Les deux familles suivantes (*IN* et *OUT*) ont une nature opposée. *IN* représente des pages qui peuvent atteindre le noyau mais qui ne peuvent pas être atteintes par lui - les arcs étant orientés. Au contraire, *OUT* inclut les documents accessibles par les pages de *SCC* mais qui n'ont pas de lien retour, comme par exemple les sites de grandes compagnies qui ne contiennent pratiquement que des liens internes à leur site Web. Enfin, les *ramifications* contiennent des pages qui ne peuvent atteindre le noyau et réciproquement ne peuvent pas être atteintes par lui. Toutes de ces familles comprennent approximativement le même nombre de pages (environ 44 millions) avec cependant un léger avantage pour le noyau qui contient quasiment 56 millions de pages.

Le diamètre maximum effectif de *SCC* a été mesuré à au moins 28 liens et le diamètre maximum du graphe complet est supérieur à 500 liens. Dans le sous-graphe de l'étude, la probabilité qu'il existe un chemin d'une source à une destination n'est seulement que de 24%. Si ce chemin existe, la longueur moyenne est d'environ 16 liens ; cependant, en

considérant le graphe non orienté, la longueur de ce chemin n'est plus que de 6 liens.

Ces analyses statistiques nous permettent de mieux comprendre l'organisation des documents d'Internet afin de rendre des algorithmes de recherche les plus efficaces possible. Actuellement, deux algorithmes d'analyse des liens contenus dans les différents documents se sont imposés grâce à leur grande efficacité. Il s'agit de *HITS* et de *PageRank*.

1.2.2 L'algorithme *HITS*

Kleinberg fut un des premiers à s'intéresser aux propriétés de connectivité du graphe représentatif d'Internet et de son apport dans la détection de la pertinence d'une page à une requête [Kleinberg, 1999]. Quelques constatations simples sont à l'origine de ses travaux dans ce domaine.

Comme nous avons pu le voir dans la section 1.2.1, l'importance d'une page peut être extraite de la structure des liens du Web. Dans cette approche, deux types de page sont identifiés en fonction de la nature de leurs connexions avec les autres documents. On retrouve d'une part les pages qui semblent être très importantes et jouent le rôle d'*autorité* sur un sujet donné et d'autre part les documents possédant un grand nombre de liens vers des pages faisant autorité sur un sujet. On distingue ainsi les pages *autorités* ayant un grand nombre de liens entrants et les pages *hubs* ayant un grand nombre de liens sortants et regroupant les autorités d'un même sujet. Le but de l'algorithme *HITS* est de déterminer les *hubs* et les *autorités* qui renforcent leurs relations mutuellement sur un sujet donné. Ainsi Kleinberg dénombre les bons hubs comme des pages pointant vers beaucoup de bonnes autorités et les bonnes autorités comme des pages pointées par beaucoup de bons hubs.

Cette dénotation de *bons hubs* et de *bonnes autorités* fait apparaître une troisième catégorie de pages ayant un grand nombre de liens entrants provenant de documents n'ayant aucune particularité. Ces pages, que nous nommons *pages indépendantes*, sont considérées comme universellement populaires et n'apportent que peu ou pas d'intérêt [Chakrabarti *et al.*, 1999]. Par exemple la page de *Google* est extrêmement référencée mais n'apporte que peu d'intérêt sur la plupart des sujets rencontrés ou une publicité aura de nombreux liens vers elle. Cette situation est illustrée dans la figure 1.6.

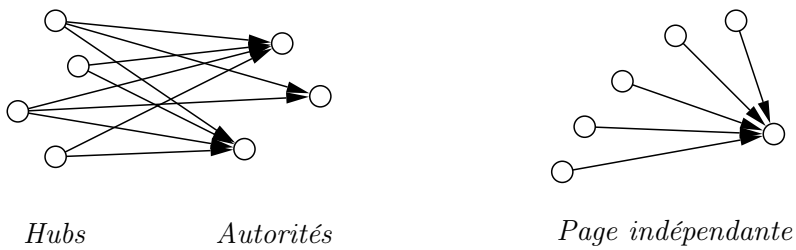


FIG. 1.6 – Répartitions des pages en *bon* Hubs, *bonnes* autorités et pages indépendantes.

Une justification intuitive de l'autorité conférée à une page en fonction de la structure des liens l'entourant peut être donnée en considérant qu'un fort taux de jugement humain entoure l'ajout d'un lien hypertexte dans un document. En quelque sorte, l'auteur du document estime que la page vers laquelle il construit un lien évoque un sujet similaire à son souhait et paraît intéressante.

Pour déterminer les hubs et les autorités d'un sujet σ donné, l'algorithme HITS se base sur un sous-graphe d'Internet S_σ qui doit répondre aux conditions suivantes :

- S_σ est relativement petit,
- S_σ est riche en pages pertinentes,
- S_σ contient la totalité (ou la plupart) des plus importantes autorités.

En gardant S_σ petit, l'application d'algorithmes non triviaux peut s'effectuer sans s'occuper du temps de calcul nécessaire à la réalisation de la tâche. Les deux derniers points nous permettent de nous assurer d'avoir de bonnes chances de déterminer les bonnes autorités correspondant à la requête σ.

Pour construire un tel graphe, l'algorithme utilise une requête de mots-clés (σ) afin de prendre en compte un petit nombre de pages - environ 200 - depuis un moteur de recherche traditionnel à base d'index inversé. Cependant, cet ensemble de pages, noté R_σ, ne contient pas nécessairement l'ensemble des autorités du sujet σ. Par exemple, il y a de fortes chances pour que les réponses à la requête *moteur de recherche* ne correspondent pas aux grands sites de moteurs de recherche : ces pages ne contiennent en effet que très rarement les mots-clés recherchés. On peut toutefois espérer que ce sous-graphe contienne des liens vers des autorités ou des hubs importants. R_σ est alors étendu en ajoutant les nœuds pointés par le sous-graphe et les nœuds pointant le sous-graphe afin d'obtenir un graphe augmenté S_σ. L'auteur affirme alors que les trois conditions requises par le sous-graphe S_σ sont respectées.

Les *bons hubs* et les *bonnes autorités* peuvent être extraits de ce graphe en donnant une définition numérique à la notion de hub et d'autorité. L'algorithme *HITS* associe un vecteur de potentiels d'autorités non négatifs $\langle x \rangle$ et un vecteur de potentiels de hubs non négatifs $\langle y \rangle$ pour toutes les pages appartenant à S_σ. Ainsi, une page p possédant un fort potentiel d'autorité x_p, respectivement un fort potentiel de hub y_p, sera vue comme étant une *bonne autorité*, respectivement un *bon hub*.

Initialement, aucune hypothèse n'est faite sur les potentiels hub et autorité de chaque document. Ainsi les vecteurs originaux $\langle x \rangle$ et $\langle y \rangle$ sont uniformes. Les poids sont ensuite mis à jour de la façon suivante : si une page est pointée par plusieurs *bons hubs*, nous incrémentons son potentiel d'autorité. Ainsi, pour une page p donnée, la valeur de son poids x_p est mise à jour par la somme des potentiels y_q de toutes les pages q pointant sur p. D'une manière symétrique, les potentiels de hub sont mis à jour en fonction des potentiels d'autorités des pages vers lesquelles notre document en cours pointe. Ces opérateurs sont décrits dans la figure 1.7.

La sortie de l'algorithme *HITS* est alors composée de deux listes ordonnées décroissantes de pages en fonction des potentiels respectifs de hub et d'autorité. L'originalité vient du fait que seule la topologie des liens entre les documents est prise en compte afin de déterminer les pages les plus pertinentes sur un sujet donné. Les expérimentations

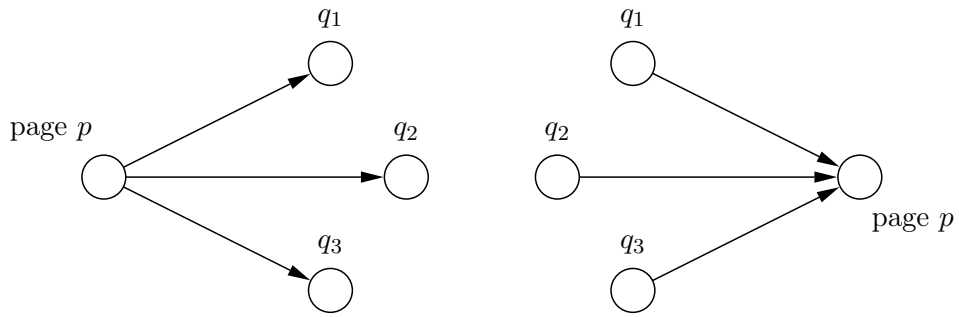


FIG. 1.7 – Opérateurs de mise à jour des accumulateurs Hub (y_p) et Autorité (x_p).

réalisées par Kleinberg montrent que sur une requête problématique pour un moteur de recherche basé uniquement sur un index inversé, les résultats obtenus sont très satisfaisants : la requête *search engine* -posée en 1999 - produit les sites *Yahoo !*, *Excite*, *Magellan*, *Lycos* et *Altavista* comme autorités bien que leur texte ne contienne pas la requête initiale.

La première grande implémentation de *HITS* fut réalisée au travers du système *ARC* et de l'algorithme *CLEVER* conçus au centre de recherche IBM Almaden et décrits dans [Chakrabarti *et al.*, 1998a]. Les auteurs ont tenté d'apporter certaines modifications en vue d'améliorer les résultats.

L'étape d'initialisation de ce dernier algorithme est essentiellement la même que celle de *HITS*. Le noyau de pages R_σ est cependant augmenté en suivant deux liens entrants et sortants afin de produire un sous-graphe S_σ plus important. La mise à jour des potentiels est elle aussi revue en prenant en compte une nouvelle pondération sur les sommes en fonction du texte des pages concernées. Ces poids sont basés sur le nombre d'apparitions des termes $n(t)$ de la requête dans les alentours de chaque lien exploré. La zone d'exploration autour de chaque lien a été déterminée de manière statistique à 50 caractères de part et d'autre du lien. Ainsi, les arcs du graphe S_σ sont dorénavant pondérés par :

$$\omega(p, q) = 1 + n(t) \tag{1.16}$$

D'autres auteurs ont tenté d'améliorer encore l'algorithme de départ, notamment [Borodin *et al.*, 2001] qui propose une correction sur le calcul des potentiels de hub. En effet, si on considère un graphe possédant $M + 1$ pages hubs et $M + 1$ pages autorités avec $M \gg 0$, et si les premiers M hubs pointent uniquement sur la première autorité, le hub restant pointant sur toutes les $M+1$ autorités, on devrait s'attendre à ce que le dernier hub obtienne un plus faible score que les autres du fait qu'il pointe vers beaucoup de *mauvaises* autorités. Cependant, l'algorithme de base fera de lui le meilleur hub. La modification proposée altère la mise à jour des hubs en leur donnant pour valeur la

moyenne des scores des autorités vers lesquelles ils pointent.

Toutes ces méthodes reposent sur un sous-graphe d'Internet afin de réduire les temps d'exécution des algorithmes. Ce sous-graphe est obtenu à partir d'une requête donnée. Le calcul de pertinence est alors réalisé pour chaque requête parvenue au moteur de recherche. Il existe cependant des méthodes indépendantes de la requête proposée par l'utilisateur qui peuvent déterminer une pertinence *absolue* des documents du réseau. La plus célèbre repose sur l'algorithme du *PageRank* [Page *et al.*, 1998, Brin et Page, 1998, Cho *et al.*, 1998] et est à l'origine du moteur de recherche *Google*.

1.2.3 L'algorithme *PageRank*

L'idée principale de cette méthode est de simuler le comportement d'un internaute naviguant de manière aléatoire sur Internet. La probabilité qu'il visite une page donnée est d'autant plus grande que cette page est pointée par beaucoup d'autres pages au travers de leurs liens hypertextes. En considérant qu'une page confère une certaine autorité à une autre page en établissant un lien vers elle, la probabilité de passage de cet internaute aléatoire sur une page indique le degré de pertinence de ce document.

Il existe deux manières d'accéder à une page Web sur Internet. On peut d'une part l'atteindre directement en connaissant son adresse et d'autre part suivre un lien hypertexte d'un autre document. Le calcul du *PageRank* - et donc de la pertinence - d'une page intègre ces deux éléments au travers d'une probabilité d. d représente en quelque sorte la probabilité que l'internaute aléatoire s'ennuie sur une page et décide de choisir une autre page au hasard. L'équation 1.17 montre la récursion permettant de calculer le *PageRank* pour tous les nœuds du graphe représentant Internet.

$$PR(p_j)_t = (1-d) + d \sum_{\substack{i=1 \\ p_i \to p_j}}^{n} \frac{PR(p_i)_{t-1}}{C(p_i)} \tag{1.17}$$

Dans cette formule, $PR(p_j)_t$ représente la valeur du *PageRank* à l'itération t pour la page p_j. $C(p_i)$ est défini comme le nombre de liens sortants de la page p_i Le paramètre d prend ses valeurs dans l'intervale $[0-1]$ et est généralement placé à $d = 0.85$ d'après des études statistiques menées par Lary Page dans [Page *et al.*, 1998]. Ce dernier paramètre permet de faire converger l'algorithme de manière plus ou moins rapide. En effet, plus d est élevé, plus l'effet de l'ajout d'un lien entrant vers une page est accru et plus celui-ci se propagera dans toutes les pages d'un même site.

Le *PageRank* forme ainsi une distribution de probabilités des pages Web. Le calcul peut s'effectuer de manière itérative et converge vers une valeur asymptotique de manière assez rapide. En effet, le calcul effectué dans [Page *et al.*, 1998] sur un graphe de 26 millions de nœuds en considérant $d = 0.85$, converge en seulement 52 itérations.

On constate dans la formule que pour chaque itération toutes les pages distribuent leur *PageRank* en fonction de leurs liens sortants. Cette distribution s'effectue de manière plus ou moins importante en fonction du nombre de liens $C(p_i)$ de chaque page p_i (distribution uniforme). Cela peut aisément s'interpréter en considérant qu'une page

possède un certain potentiel de pertinence et que l'auteur de la page donne une certaine crédibilité - son *PageRank* - aux pages pour lesquelles il inscrit un lien.

La formule initiale suppose que la probabilité de *sauter* sur n'importe quelle page est uniforme. Mais une des plus importantes variations de cet algorithme consiste à attribuer une distribution de probabilité non uniforme afin de mieux caractériser la réalité d'Internet [Brin *et al.*, 1998]. La première partie de l'équation 1.17 devient alors $(1-d) \cdot E_{p_j}$.

L'équation 1.17 peut être reformulée de manière matricielle. Pour cela, on considère deux vecteurs colonne PR et E correspondant respectivement au rang de chaque page et à la loi de distribution de l'ensemble des nœuds du graphe - loi de distribution uniforme dans la formule initiale mais personnalisable comme vu par exemple dans le paragraphe précédent -, et une matrice A telle que l'entrée $A[p,q] = 0$ s'il n'existe pas de lien sortant de la page q vers la page p et $A[p,q] = \frac{1}{C(p)}$ sinon. A est une matrice sous-stochastique. Cette équation prend ainsi la forme suivante :

$$PR = (1-d)E + d \cdot A^t \cdot PR \tag{1.18}$$

Où A^t désigne la transposée de la matrice A. La sommation de l'équation 1.17 peut être représentée par la multiplication matricielle $A^t \cdot PR$. Ainsi, PR correspond au vecteur propre principal de la matrice normalisée des liens du Web.

Afin d'améliorer les performances de cet algorithme, une autre variation a été introduite conduisant à modifier, dans l'évaluation du *PageRank* d'une page, l'importance de l'apport de chaque lien au calcul final. Des heuristiques de modifications du poids de chaque lien ont notamment été étudiées dans [Chakrabarti *et al.*, 1998a] et [Bharat et Henzinger, 1998]. Ces heuristiques sont principalement basées sur l'analyse des textes entourant et présents sur les différents liens hypertextes contenus dans les pages Web et sur l'utilisation de mesures de similarités basées sur le modèle vectoriel et la mesure *tf*idf* (voir section 1.1.3).

Deux problèmes avec l'approche du *PageRank* sont soulignés par Kleinberg dans [Kleinberg, 1999]. Premièrement, plusieurs pages peuvent être considérées comme des *autorités* particulièrement importantes sur des sujets donnés mais ne contenant pas les termes de la requête correspondante (voir section 1.2.2). Cela pose un problème de par la méthode de sélection des pages utilisée et décrite dans cette section : la sélection des réponses pertinentes se fait simplement à l'aide d'un index inversé et le tri de ces pages est effectué grâce au score obtenu par le *PageRank*. Les solutions obtenues contiennent alors forcément les termes de la requête.

Un autre problème est souligné concernant les pages qui ont un nombre de liens entrants très important comme par exemple *www.yahoo.com*. Ces pages auront un *PageRank* très élevé. Cependant, si la requête proposée contient un mot compris dans ces pages, en reprenant l'exemple précédant avec le terme *Automobiles*, ces sites vont

être retournés à l'utilisateur en tête de liste bien qu'ils ne fassent pas du tout autorité sur le sujet donné. Kleinberg conclut d'ailleurs en disant que l'utilisation unique de la structure des liens entrants ne permet pas d'obtenir un équilibre correct entre la notion de popularité et la notion de pertinence.

Afin de corriger ces éventuels problèmes, plusieurs auteurs ont tenté d'effectuer une hybridation entre les algorithmes *PageRank* et *HITS*. Citons par exemple *HubRank* [Chirita *et al.*, 2003] qui modifie la matrice de personnalisation de distribution de probabilité E de l'équation 1.18 en prenant en compte à la fois la distribution des liens sortants et des liens entrants du graphe de connexion des pages Web. Le surfer aléatoire va alors préférer aller sur une page ayant un fort degré de liens sortants lorsqu'il s'ennuie - avec la probabilité $(1 - d)$.

Un autre exemple d'amélioration intéressante est donné dans [Ng *et al.*, 2001]. Les auteurs ont cherché à combiner les avantages de l'algorithme *HITS* avec l'algorithme *PageRank*. Il utilise la notion de parcours aléatoire du graphe d'Internet présent dans l'algorithme *HITS* permettant de détecter les pages aux caractéristiques de *hubs* et d'*autorités* avec l'initialisation du surfeur aléatoire présent dans la première partie de l'équation du *PageRank* 1.17. Les résultats obtenus par les auteurs montrent alors que ce nouvel algorithme est plus robuste aux perturbations rencontrées dans les différents graphes de test.

1.2.4 L'algorithme *SALSA*

Un algorithme alternatif, *SALSA*, a été proposé par Lempel et Moran en 2000 [Lempel et Moran, 2000]. Son but est similaire à celui de l'algorithme *HITS* décrit dans la section 1.2.2 : il cherche à déterminer les meilleurs pages correspondant à un sujet donné en les caractérisant en *hubs* et *autorités*.

La méthode choisie par les auteurs pour résoudre ce problème est toutefois différente de celle utilisée par Kleinberg dans l'algorithme *HITS*. Elle consiste à parcourir au hasard les nœuds du graphe d'Internet en suivant les liens les reliant avec une distribution de probabilité uniforme. Les pages les plus visitées correspondent alors aux solutions recherchées. Intuitivement, les pages faisant office d'*autorités* sur un sujet donné sont visibles - liées - depuis beaucoup de pages dans le sous-graphe étudié. Ainsi, en parcourant le graphe au hasard des liens, la probabilité de visiter une page *autorité* est grande. Cette idée de détermination stochastique de l'intérêt d'une page se rapproche de celle utilisée par Brin et Page dans le calcul du *PageRank* mais diffère dans la séparation des résultats en *hubs* et *autorités* pour un domaine donné. La détermination des probabilités d'appartenance de chaque page du graphe à l'une des deux catégories possibles se fait à l'aide de chaînes de Markov.

La première étape consiste à construire un sous-graphe d'Internet correspondant à une requête donnée, sur lequel s'effectuera la recherche de pages pertinentes. Cette construction se base sur celle utilisée dans *HITS* et requiert les mêmes propriétés que celles décrites dans la section 1.2.2.

Afin de déterminer le potentiel de *hub* et d'*autorité* de chaque page, il faut déterminer la probabilité de visite de ces pages en suivant les liens du graphe de manière aléatoire. Le parcours des arcs dans le sens initial nous permet de déterminer les potentiels d'*autorités* des pages : on mesure ici la probabilité qu'une page soit pointée par beaucoup d'autres pages. Le parcours du graphe en suivant les arcs dans leur sens inverse nous donne les potentiels de *hub* de chaque page : on mesure, dans ce cas, la probabilité qu'une page pointe vers beaucoup d'autres pages.

Deux chaînes de Markov sont alors analysées : une chaîne de *hubs* et une chaîne d'*autorités*. Les états de transitions de ces chaînes sont générés en effectuant le parcours aléatoire du graphe. Mais, contrairement à un parcours classique des liens, deux liens hypertextes sont traversés : 1) en choisissant selon une probabilité uniforme d'aller sur une page pointée par la page actuelle, et 2) en choisissant selon une probabilité uniforme d'aller sur une page pointant vers la page actuelle.

Les potentiels d'*autorités* sont alors définis comme étant la distribution stationnaire de la chaîne de Markov effectuant d'abord un pas 1) puis un pas 2), tandis que les potentiels de *hubs* sont définis comme la distribution stationnaire de la chaînes effectuant d'abord un pas 2) puis un pas 1).

Formellement, les deux matrices de transition des deux chaînes de Markov sont définies ainsi :

1. La matrice déterminant les potentiels de *hub* $\tilde{H}$,

$$\tilde{h}_{i,j} = \sum_{k:k\in S(i)\cap S(j)} \frac{1}{|S(i)|} \cdot \frac{1}{|E(k)|} \tag{1.19}$$

2. La matrice déterminant les potentiels d'*autorité* $\tilde{A}$,

$$\tilde{a}_{i,j} = \sum_{k:k\in E(i)\cap E(j)} \frac{1}{|E(i)|} \cdot \frac{1}{|S(k)|} \tag{1.20}$$

Dans ces formules, $E(i)$ décrit tous les nœuds du graphe pointant vers le nœud i, et donc les pages que nous pouvons atteindre depuis la page i en suivant un lien dans le sens inverse. $S(i)$ décrit tous les nœuds que nous pouvons atteindre depuis le nœud i en suivant un lien de i.

Une probabilité de transition $\tilde{a}_{i,j} > 0$ indique qu'une certaine page k pointe à la fois vers les pages i et j et que, de plus, la page j peut être atteinte depuis la page i en deux pas : en parcourant le lien de la page k vers i dans le sens inverse et en suivant ensuite le lien de la page k vers la page j.

Les auteurs ont prouvé que la distribution stationnaire $h = (h_1, h_2, \ldots, h_N)$ de la chaîne de Markov satisfait $h_i = \frac{|S(i)|}{|S|}$ où $S = \cup_i S(i)$ est l'ensemble des liens sortants ; et que la distribution stationnaire $a = (a_1, a_2, \ldots, a_N)$ de la chaîne de Markov satisfait $a_i = \frac{|E(i)|}{|E|}$ où $B = \cup_i E(i)$ est l'ensemble des liens entrants.

Cet algorithme n'établit pas les potentiels de *hub* et d'*autorité* de la même manière que l'algorithme *HITS* dans lequel la structure se renforce mutuellement. En effet,

puisque $a_i = \frac{|E(i)|}{|E|}$, l'autorité relative du nœud i est déterminée non pas depuis des liens globaux, mais depuis les liens locaux à ce nœud.

Le renforcement mutuel par la structure des liens du graphe de l'algorithme *HITS* fait qu'il pose problème dans certains cas, notamment ceux identifiés par l'effet *TKC* (Tightly-Knit Community). Cet effet apparaît lorsqu'une communauté de pages obtient un très bon score par les algorithmes d'établissement de pertinence par mesures topologiques, bien qu'elles ne fassent pas autorité sur le sujet donné. Ces communautés sont petites mais très fortement interconnectées entre elles. Les auteurs ont toutefois prouvé que bien que l'algorithme *HITS* soit affecté par ce problème, *SALSA* arrive à bien évaluer ces communautés.

L'algorithme *SALSA* a été généralisé par Mendelzon [Mendelzon et Rafiei, 2000]. Il s'agit en fait d'une hybridation de cet algorithme et de l'algorithme du *Pagerank* afin d'établir la réputation de chaque page. Les auteurs ont adapté le *facteur de zap* de Brin et Page en considérant qu'avec une probabilité d, l'algorithme de Mendelzon *saute* sur une nouvelle page suivant une loi de probabilité uniforme et avec une probabilité $1-d$, il effectue une itération de l'algorithme *SALSA*.

1.2.5 L'algorithme *PHITS*

D'autres approches de détermination de *hubs* et *autorités* ont également été testées. Cohn et Chang ont proposé un algorithme statistique afin de déterminer ces deux catégories [Cohn et Chang, 2000].

Le modèle que les auteurs ont construit tente d'expliquer deux types de variable, les citations c d'un document d en fonction d'un petit nombre de variables communes z qui sont appelées les *aspects* ou les *facteurs*. Ces variables communes peuvent être considérées comme des sujets ou des communautés de pages.

Le modèle peut alors être décrit statistiquement : un document $d \in D$ est généré avec une probabilité $P(d)$, le facteur, ou sujet $z \in Z$ associé à d est choisi en fonction d'une probabilité $P(z|d)$, et étant donné ce facteur, des citations $c \in C$ sont générées en fonction de la probabilité $P(c|z)$.

La probabilité de chaque paire (document, citation) (d, c) est alors décrite par :

$$P(d,c) = P(d)P(c|d) \tag{1.21}$$

$$P(c|d) = \sum_z P(c|z)P(z|d) \tag{1.22}$$

En considérant la matrice A représentant les paires (document, citation) décrites dans la section 1.2.3, où l'entrée $A[i,j]$ est non nulle si le document i possède un lien vers le document j, la probabilité de la matrice de citation A est la suivante :

$$L(A) = \prod_{(d,c)\in A} P(d,c) \tag{1.23}$$

Le problème consiste alors à trouver les valeurs de $P(d)$, $P(z|d)$ et de $P(c|z)$ qui maximisent la fonction de probabilité $L(A)$ des données observées. Afin de résoudre ce nouveau problème, les auteurs proposent d'utiliser l'algorithme *EM* de Dempster [Dempster *et al.*, 1977].

Ce modèle entièrement probabiliste a l'avantage d'apporter plus d'informations que le modèle utilisé par l'algorithme *HITS*. Une analogie peut être toutefois faite en considérant les *autorités* sur un sujet donné comme la probabilité conditionnelle $P(c|z)$ qui indique de quelle manière un document c est cité depuis une communauté z. Mais d'autres informations peuvent être extraites du modèle comme par exemple la probabilité $P(z|c)$ qui nous permet de connaître la communauté à laquelle appartient un document c donné, ou encore la découverte des documents caractéristiques d'une communauté donnée en déterminant le produit $P(z|c) \cdot P(c|z)$.

Malgré tout, cet algorithme impose de connaître à l'avance le nombre de facteurs z à prendre en considération. De plus, il est possible que l'algorithme *EM* puisse rester bloqué dans un maximum local et compromettre la convergence vers le maximum global correspondant à la solution du problème.

Comme nous avons pu le voir, les techniques de recherche d'informations sont très variées et prennent en compte des éléments très différents. À l'heure actuelle, l'analyse de la topologie d'Internet est un élément prépondérant dans la détection de la pertinence d'un document. Ce système permet en effet de récupérer un certain jugement humain inscrit dans les liens hypertextes : lorsqu'une personne décide d'inscrire un lien hypertexte dans une page Web, elle considère que le document pointé par ce lien apporte une information utile.

1.2.6 Les méta-moteurs

Les méta-moteurs présentent des stratégies de recherche beaucoup plus hétérogènes que ce que l'on peut trouver dans les moteurs de recherche classiques. Cependant, tous ont pour point commun d'utiliser les résultats produits par ces mêmes moteurs de recherche.

La plupart de ces outils ne fait que trier les liens récupérés de plusieurs sources en utilisant ses propres algorithmes de détection de pertinence. Dans ce cas, leur but est de tirer parti de la complémentarité et de la spécialisation de plusieurs outils de recherche. Dans ce cas ces outils se distinguent par les fonctions de tri utilisées qui sont propres à chaque méta-moteur.

Cependant, il existe d'autres méta-moteurs que l'on peut qualifier de plus évolués. Ils ne se contentent pas de compiler les résultats d'autres moteurs de recherche mais parcourent également Internet à la recherche de documents pertinents. Il n'existe pas dans ce cas d'organisation type et chacun utilise ses propres stratégies afin de se distinguer des autres méta-moteurs.

Nous examinerons en détail dans la section 2.4.2 du chapitre 2 un méta-moteur

particulier : *InfoSpiders*. Il a la particularité d'utiliser plusieurs agents de recherche spécialisés et capables d'apprendre en fonction des souhaits de l'utilisateur.

D'autres ressemblent plus à des partages de ressources en regroupant des communautés d'utilisateurs. Par exemple, *AntWorld* [Meñkov *et al.*, 2000] est un outil qui prend en compte le jugement que portent des utilisateurs sur les documents qu'ils parcourent. Pour chaque page qu'il juge intéressante, un utilisateur donne les mots-clés correspondant à la page ainsi qu'un avis documenté sur celle-ci. Une base de données recense et centralise tous les avis. Si la communauté est importante, le nombre de documents accessibles devient significatif. Les résultats obtenus par l'intermédiaire d'un tel moteur sont alors très pertinents car classés par des experts.

Dans un moteur de recherche, les systèmes d'interrogation et de sélection des pages jouent un grand rôle. Mais l'énorme quantité de données à traiter et l'évolution permanente de l'information qui augmente sans cesse imposent d'avoir une architecture du système de recherche particulièrement élaborée.

1.3 Architectures des moteurs de recherche

Les architectures utilisées dans les systèmes de recherche d'information ont beaucoup évolué au cours de la dernière décennie, passant de systèmes monolithiques particulièrement peu adaptatifs à des systèmes fortement distribués s'ajustant en permanence à la quantité d'information à analyser. Cette évolution a été rendue nécessaire - et même obligatoire - car le nombre d'internautes ne cesse de progresser. De plus, les moteurs de recherche sont devenus le point de départ de beaucoup de sessions de navigation sur Internet et il a donc fallu trouver des solutions pour répondre en des temps raisonnables aux innombrables requêtes formulées à chaque instant.

1.3.1 Architecture générale des premiers moteurs de recherche

L'architecture originale utilisée par *Altavista* [Altavista] représente la première catégorie de systèmes. Son fonctionnement est décrit dans la figure 1.8. Il s'agit d'une architecture très simple qui se divise en deux parties distinctes. On retrouve d'une part un *crawler* et d'autre part l'interface d'interrogation du moteur de recherche et le système d'analyse des requêtes proposés par les utilisateurs du système. Le *crawler* peut être considéré comme un robot chargé de rapatrier tous les documents Web contenus sur Internet dans un index centralisé en suivant les liens hypertextes rencontrés dans les pages analysées [Baeza-Yates et Ribeiro-Neto, 1999].

Le cœur du système repose sur un index inversé permettant d'associer des mots à un ou plusieurs documents. La demande de l'utilisateur est traitée en interrogeant l'index inversé pour connaître les documents dans lesquels apparaissent le plus souvent les mots de la requête.

Ce système est très peu évolutif et nécessite un matériel très avancé. Par exemple, en 1998, *Altavista* utilisait un système comprenant 20 processeurs auxquels étaient alloués 130 Go de mémoire vive et 500 Go de disque dur. Dans ce modèle, le moteur de requête

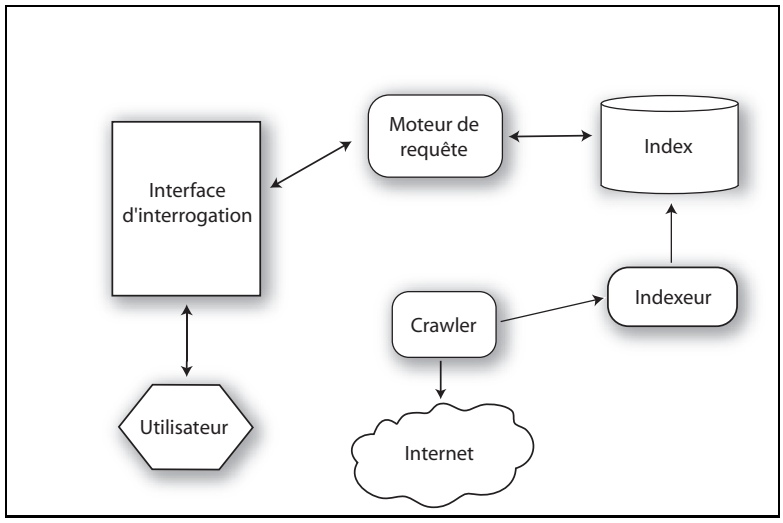


FIG. 1.8 – Architecture originale du moteur de recherche *Altavista*.

consommait à lui seul 75% des ressources mises à disposition par le système. Cette architecture monolithique ne permet pas de faire évoluer le système dynamiquement. Or, Internet est en perpétuelle expansion ce qui rend ce système obsolète aujourd'hui.

Depuis ces années - et surtout depuis l'apparition du moteur de recherche *Google* - l'architecture d'*Altavista* a évolué vers les modèles décrits dans la suite de cette section.

1.3.2 Vers un modèle distribué et adaptatif

Des variantes de l'architecture précédente, basées sur le modèle indexeur-crawler, ont été imaginées afin de gommer les défauts inhérents à sa conception. L'une d'entre elle, appelée *Harvest* s'est révélée très innovante en matière de distribution des ressources. Cette architecture est décrite également dans [Baeza-Yates et Ribeiro-Neto, 1999] et a été utilisée par de nombreux organismes comme la NASA, l'Académie des Sciences Nationale des Etats-Unis d'Amérique.

Comme le montre la figure 1.9, cette architecture se développe autour de deux composantes principales : le récolteur et le broker. Chacun de ces éléments a un rôle particulier à jouer dans la chaîne de traitement du moteur de recherche. Le récolteur est chargé de collecter et d'extraire périodiquement des informations d'indexation - textes, images - depuis plusieurs sites Web. Le broker, quant à lui, fournit le mécanisme d'indexation et l'interface d'interrogation sur les données amassées par le récolteur. On retrouve ici, le mécanisme indexeur-crawler identifié dans la section précédente. Cependant, plusieurs brokers et plusieurs récolteurs peuvent communiquer ensemble, chacun se spécialisant dans un domaine précis. Lorsqu'une requête est émise sur un broker dont le domaine traité ne correspond pas à ses capacités, celui-ci transmet la requête à une autre entité capable de la gérer.

C'est un système totalement adaptatif dans lequel il est possible de configurer les

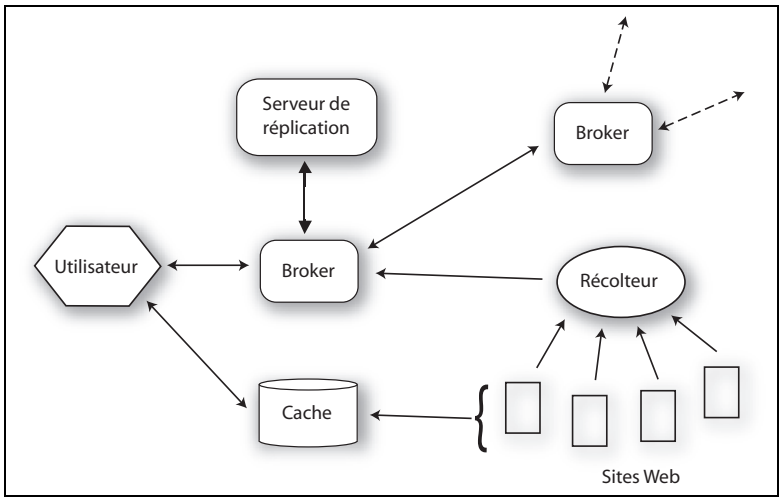


FIG. 1.9 – Architecture du système *Harvest*.

brokers et les récolteurs de manière à répartir le besoin en ressources sur un ou plusieurs domaines particuliers. Un système de réplication permet de plus de garantir une qualité de service relativement fiable.

Lancé en 1998, *Google* a révolutionné le monde des moteurs de recherche. Son système de classement des résultats (présenté dans la section 1.2.3) mais aussi sa grande rapidité d'exécution des requêtes l'ont rapidement placé en tête des outils de recherche d'information sur Internet.

1.3.3 Architecture moderne d'un moteur de recherche

L'architecture du moteur de recherche *Google* est certainement une des plus efficaces actuellement. Elle ne repose pas sur un système monolithique mais sur un grand nombre de machines classiques coopérant ensemble. La figure 1.10 montre l'organisation des différents éléments qui composent cette architecture.

Ce système peut se décomposer en plusieurs parties comprenant un sous-système d'exploration d'Internet, un indexeur, un analyseur de la topologie d'Internet formée par les liens hypertextes et un sous-système de présentation et d'exécution de requêtes.

Un serveur d'URL garde la mémoire des liens des pages à visiter. Des robots chargés d'explorer le Web récupèrent ces liens afin de télécharger les documents correspondant et les stocker dans une base de données recensant la totalité des pages indexées. Cette opération est réalisée continuellement et alimente et met à jour en permanence la base de documents du moteur. Périodiquement, cette base est analysée afin de réaliser un index inversé reliant des termes aux documents les contenant. D'autres informations sur les termes sont extraites comme leur position dans le document, la taille de la police utilisée ou sa fonte. Cet index inversé est distribué sur une multitude d'ordinateurs désignés par le terme *barrel* sur le schéma 1.10.

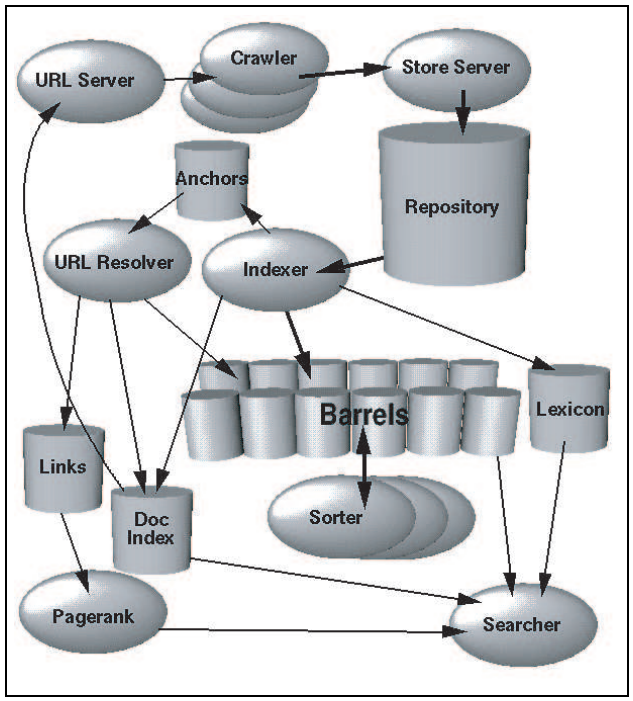


FIG. 1.10 – Architecture du moteur de recherche *Google*.

Cette analyse permet également d'extraire tous les liens hypertextes des documents rencontrés afin d'alimenter le serveur d'URL. Cette base de liens est utilisée afin de calculer le *PageRank* (voir section 1.2.3) permettant de trier les documents de l'index par pertinence décroissante.

Cette architecture est distribuée sur un grand nombre de machines standards - environ 15 000 en 2001 - et permet d'obtenir des temps de réponse inférieurs à la seconde pour une requête donnée. C'est actuellement le moteur de recherche le plus utilisé et il a montré, depuis son lancement en 1998 une très grande adaptabilité à l'évolution de la taille d'Internet jusqu'à indexer en juin 2004 près de 4,3 milliards de pages Web.

Les trois architectures de moteurs de recherche présentées montrent une évolution constante dans la rapidité et l'adaptabilité des outils de recherche s'adressant à un grand nombre d'utilisateurs. Ce type de système est devenu le point de départ d'un grand nombre de sessions de navigation sur Internet et a du s'adapter à l'afflux de requêtes toujours plus important.

Mais aussi fiables que peuvent l'être ces techniques, un outil de recherche doit présenter les résultats à un utilisateur de manière organisée. En effet, devant le nombre impressionnant de liens retournés par un moteur de recherche pour une requête donnée, on peut rapidement se sentir submergé. Par conséquent, il est nécessaire d'étudier la

meilleure manière de présenter les résultats d'une recherche afin de faciliter leur lecture et leur compréhension.

1.4 Visualisation et présentation des résultats

1.4.1 Généralités sur la visualisation de résultats dans les moteurs de recherche

L'affichage des résultats d'une requête est l'un des problèmes majeurs pour un moteur de recherche. En effet, que les résultats soient pertinents ou non, si l'utilisateur ne comprend pas le plus rapidement possible la manière dont s'organisent les éléments de réponses, ces derniers ne lui seront d'aucune utilité.

Depuis ces dernières années, les moteurs de recherche, ou plus exactement les méta moteurs de recherche commerciaux en ont pris conscience et proposent des interfaces de présentation des résultats de plus en plus interactives et visuelles, ne se contentant plus d'une simple liste de réponses. Le méta moteur de recherche *Kartoo* [Kartoo] a été l'un des pionniers dans ce domaine en présentant pour la première fois ses résultats sous forme d'une carte indiquant les principales relations entre les différents résultats de la recherche comme le montre l'illustration 1.11.

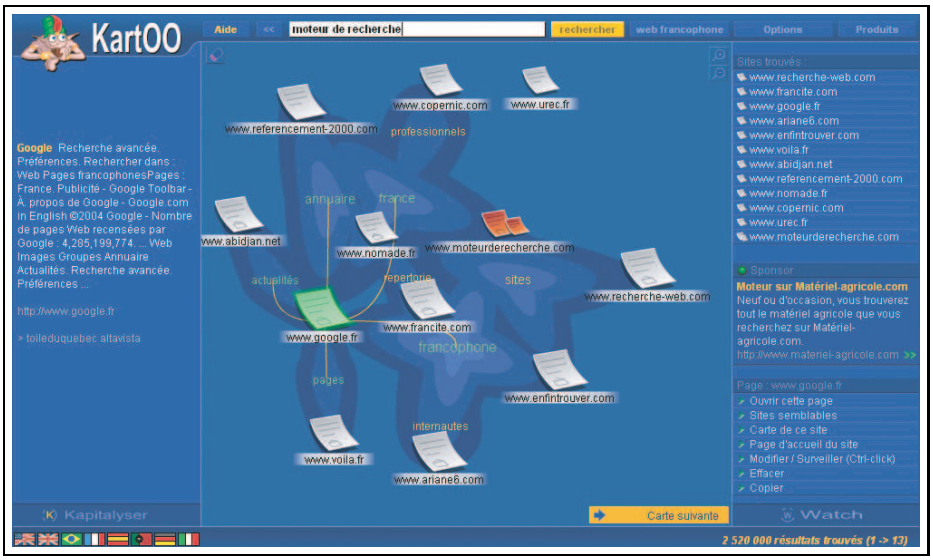


FIG. 1.11 – Exemple d'affichage de résultats par le méta moteur de recherche *Kartoo*.

Cependant, comme le faisait déjà remarquer Washburne au début du siècle dernier [Washburne, 1927], il n'existe pas de visualisation efficace pour toutes les données et tous les utilisateurs. Il existe malgré tout plusieurs méthodes apportant de l'intérêt dans l'affichage de résultats de moteurs de recherche de la plus simple à la plus sophistiquée. Et comme le souligne Cugini dans [Cugini, 2000], toute technique de visualisation requiert de l'apprentissage de la part de l'utilisateur afin qu'il puisse l'utiliser efficace-

ment. Chaque technique peut ainsi s'adapter différemment en fonction du public visé dans l'application de recherche d'information.

Cela implique une plus lente évolution des outils de recherche destinés au plus grand nombre de personnes tels les moteurs de recherche classiques. En effet, leur interface de présentation des résultats s'est très peu améliorée au cours des dix dernières années et l'évolution la plus importante est à mettre à l'actif des méta moteurs de recherche s'adressant à un public beaucoup plus restreint.

Dans cette optique, et fort de l'expérience acquise grâce un travail effectué en collaboration avec le moteur de recherche *Altavista*, le concepteur du système *Exalead* [Exalead] a conçu son interface dans l'optique d'expliquer à l'utilisateur de quelle manière sont retournés les résultats correspondant à la requête demandée. À chaque étape de la recherche, des informations lui sont transmises et le guident afin d'affiner au mieux son souhait de recherche.

Dans [Mann, 1999], l'apport de la visualisation pour l'utilisateur d'un système d'information est séparé en trois niveaux distincts. Pour chaque niveau, le but recherché est différent et il faut donc adapter les techniques utilisées pour chacun d'eux.

Dans la représentation de l'ensemble des résultats, il est intéressant d'avoir une vue générale des documents retournés. Il peut être bon, par exemple, d'indiquer les différentes catégories de documents présents ou également l'existence de tendances ou de groupements possibles de résultats entre eux. Dans ce but, on utilise généralement des visualisations à base de *scatterplot* ou des métaphores à base de représentations par cartes.

Le deuxième niveau identifié par Mann concerne la visualisation de la structure d'un site Web, d'une partie d'Internet ou du chemin suivi par l'utilisateur afin de trouver ses résultats. Elle apporte elle-aussi un intérêt supplémentaire dans la compréhension des éléments retournés par le système s'information. L'utilisateur peut être intéressé par un site dans lequel un certain nombre de documents pertinents ont été trouvés, l'organisation même du site peut apporter une information pertinente. Les techniques employées dans ce cas sont basées sur des visualisations de graphes ou encore sur des techniques de zoom permettant de se focaliser à un endroit précis d'une vaste collection de données.

Enfin, à un niveau de détail plus important, le document lui-même peut être représenté de manière plus compréhensible pour l'utilisateur. Les besoins de visualisation concernent alors les techniques de production d'images réduites des documents, permettant d'avoir un aperçu graphique rapide de la page - cette technique est présentée dans [Kaugars, 1998] et [Ogden *et al.*, 1998] - ou de barres d'indications de pertinence.

D'autre part, Mann avance qu'il existe principalement quatre facteurs influençant l'utilité relative d'une visualisation donnée. Ces facteurs sont regroupés sous le terme de l'environnement des 4 T : *Target user group*, *Type and number of data*, *Task to be done* et *Technical possibilities*.

Une interface visuelle doit ainsi être adaptée au public visé, ce qui conforte la plus

grande adaptabilité que peut offrir un méta moteur de recherche vis-à-vis d'un moteur de recherche classique, à vocation beaucoup plus étendue. Le type de donnée à afficher est également prépondérant dans le choix de la représentation graphique : une visualisation adaptée au choix de quelques documents ne le sera plus forcément lorsque le nombre de documents deviendra plus important. La tâche à accomplir influence grandement l'efficacité d'une représentation particulière (voir plus particulièrement dans la section 1.4.3 les différentes représentations possible des résultats en trois dimensions en fonction de l'objectif visé : classification ou liens entre les résultats par exemple). Et pour terminer, une visualisation doit forcément être adaptée à l'environnement et aux possibilités techniques de présentation des résultats ; une interface à base de document HTML n'aura évidement pas les mêmes possibilités d'affichage que l'interface d'une application autonome.

Nous présenterons dans la suite différents systèmes de visualisation de résultats en commençant par les représentations les plus classiques à base de texte dans la section 1.4.2. Nous examinerons notamment dans la section 1.4.3 des systèmes de visualisation beaucoup plus complexes basés sur des représentations graphiques en deux ou trois dimensions qui acquièrent de plus en plus d'importance dans les systèmes de gestion d'informations que sont les métas moteurs de recherche.

Nous nous inspirerons de ces différentes applications pour concevoir un système de visualisation de résultats propre à notre outil de recherche d'information. Le chapitre 8 présentera l'ensemble des opérations mises en œuvre pour simplifier la vision des résultats de l'utilisateur.

1.4.2 Présentation par liste de réponses et classification

Traditionnellement, les résultats d'un moteur de recherche se présentent sous la forme d'une liste de liens accompagnés d'un résumé décrivant le contenu de chaque page (voir figure 1.12). Les liens présentés à l'utilisateur sont triés en fonction d'un score de pertinence attribué par des algorithmes de recherche comme ceux décrits dans la section 1.2.

Cependant, ce type de présentation très simple oblige l'utilisateur à ne s'intéresser qu'aux tout premiers résultats comme l'ont confirmé toutes les études statistiques réalisées sur l'utilisation des plus grands moteurs de recherche (voir section 1.1.5). Il est toutefois possible de rajouter des informations à ces résultats afin de permettre à l'utilisateur de cerner plus facilement le contenu des documents qui lui sont retournés.

1.4.2.1 Ajout d'information dans les listes de résultats

Afin de mieux repérer les résultats, certains auteurs ont rajouté pour chaque document retourné une barre de visualisation caractérisant les critères qui ont permis de sélectionner cette page comme étant pertinente à la requête. Cette idée n'est pas nouvelle et une des premières applications à la présentation de résultats de moteurs de recherche peut être attribuée à Hearst dans [Hearst, 1994, Hearst, 1995]. Un exemple

FIG. 1.12 – Exemple d'affichage de résultats sous forme de liste par le moteur de recherche *Google*.

d'une telle visualisation est donné dans les figures 1.13 *a)* et *b)*.

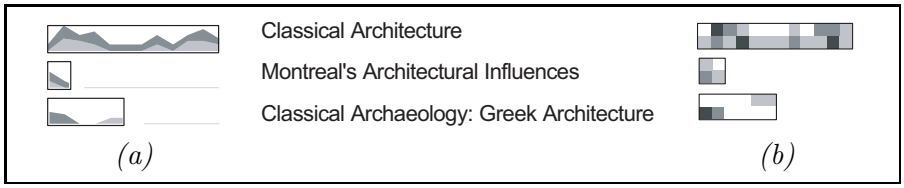


FIG. 1.13 – Exemple d'affichage de résultats sous forme de liste avec utilisation de barres de visualisations *a)* sous forme de courbe *b)* et sous forme de damier.

Cette illustration représente les résultats obtenus par l'utilisation des deux mots-clés *architecture* et *classical*. Chaque barre de visualisation représente la disposition des différents mots-clés recherchés à l'intérieur des documents retournés. La longueur des éléments est liée à la taille réelle de la page correspondante. Dans la figure 1.13 *a)*, les mots-clés recherchés sont associés à des couleurs particulières et les courbes indiquent la densité de présence de chacun d'eux dans les différentes parties des documents. Cette représentation donne une visualisation rapide de la fréquence d'apparition des éléments recherchés dans les résultats retournés ainsi que l'organisation générale des pages correspondantes.

Chaque ligne des barres de la figure 1.13 *b)* indique également la proportion des mots-clés de la requête en fonction de la zone de texte concernée. L'intensité de la couleur de chaque point indique la fréquence d'apparition du mot-clé à cet endroit du document, une couleur sombre correspondant à la présence d'un grand nombre de termes.

L'utilisation de barres d'indications peut aussi apparaître sous forme *éclatée*. La technique utilisée dans la figure 1.14 permet d'effectuer un tri rapide des résultats en fonction de la fréquence d'apparition de chaque mot-clé dans les documents retournés. Chaque document est divisé en trois barres correspondant respectivement à la pertinence générale mesurée, à la présence du mot *classical* et du mot *architecture* dans le document.

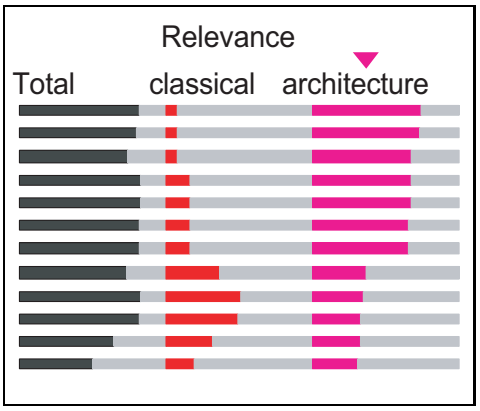


FIG. 1.14 – Représentation de résultats sous forme de liste triée par la fréquence d'apparition des mots de la requête.

D'autre part, ce type de barre est utilisé par le moteur de recherche *Google* afin d'indiquer visuellement à l'utilisateur la pertinence d'un document en fonction du *PageRank* (voir section 1.2.3) qui lui est associé.

Outre ces systèmes de visualisation, l'organisation elle-même des résultats permet de faire gagner beaucoup de temps lors de l'examen des réponses. Certaines de ces techniques ne requièrent pas beaucoup d'effort au système de recherche. Par exemple, dans [Brin et Page, 1998], les pages retournées appartenant à un même site Web sont regroupées et soulignées par un retrait de la liste des réponses. La perception du nombre de documents traitant le sujet recherché par un site est ainsi facilitée. Des systèmes de classification plus complexes ont été mis au point par plusieurs auteurs afin de simplifier l'analyse des résultats fournis.

L'analyse du comportement des utilisateurs apporte beaucoup d'informations sur la relative efficacité de ces systèmes. Les études menées dans [Drori et Alon, 2002] et [Chen et Dumais, 2000] ont cherché à caractériser les paramètres et les différentes techniques permettant d'améliorer la qualité des affichages de résultats de type liste de réponses. Elles nous donnent quelques éléments de réponse sur la manière d'afficher des résultats sous cette forme et les différents éléments additionnels à prévoir pour faciliter la découverte d'information.

Quatre affichages de listes ont été testés afin de déterminer l'apport effectif de chacun d'eux sur plusieurs plans : 1) le temps de recherche perçu par l'utilisateur pour répondre à ces questions, 2) l'importance des éléments affichés, 3) les paramètres influençant le plus les choix de l'utilisateur et 4) la confiance accordée par l'utilisateur aux résultats retournés.

Les affichages testés comportent des listes de réponses avec un résumé correspondant aux premières lignes des documents ou à des extraits dans lesquels figurent les mots de la requête d'interrogation du système. À cela s'ajoute l'affichage d'une catégorie

d'appartenance du document pré-établie.

Les résultats obtenus indiquent que les systèmes regroupant les documents par catégories apportent un réel avantage aux utilisateurs. Ceux-ci obtiennent les réponses à leurs questions initiales de manière beaucoup plus rapide et ont plus facilement confiance dans les résultats qu'ils ont pu extraire. Toutes les études montrent également clairement qu'un résumé doit accompagner chaque document retourné par le système d'information. Ce résumé ne doit comporter que quelques lignes et faire apparaître les mots-clés de la requête. Cette constatation conforte aisément les résultats obtenus par Drori dans [Drori, 2000].

Par ailleurs, l'étude comparative menée par Chen sur l'impact d'une présentation de résultats sous forme de catégories plutôt que sous forme d'une simple liste indique clairement que les utilisateurs préfèrent un pré-traitement des documents retournés afin de les regrouper [Chen et Dumais, 2000]. La représentation catégorielle mène ainsi à dépenser moitié moins de temps à rechercher l'information désirée parmi les résultats.

1.4.2.2 Regroupement des résultats en catégories

La catégorisation des documents est ainsi perçue comme un point clé de l'analyse des résultats. La littérature abordant ce sujet est assez importante et des applications se sont spécialisées dans cette technique pour l'affichage de résultats de recherche depuis quasiment l'apparition des systèmes d'information à grande échelle sur Internet. Ainsi, dès 1993, Allen a utilisé la Classification Ascendante Hiérarchique (CAH) afin de regrouper les documents similaires dans une base encyclopédique [Allen *et al.*, 1993]. Son système utilise le modèle vectoriel de représentation des documents afin de pouvoir appliquer la mesure *cosine* (voir section 1.1.3). Cette modélisation permet de déterminer une matrice de similarité de l'ensemble de son corpus de résultats. L'interface proposée à l'utilisateur se compose alors du dendogramme résultat de la CAH dans lequel il est possible de se déplacer en visualisant le document en cours.

Depuis, de nombreux systèmes de classification ont vu le jour. Ils peuvent être supervisés ou non, capables de déterminer automatiquement des catégories ou d'utiliser des groupes prédéfinis au moment de l'indexation. Mais on peut se poser la question de l'avantage de traiter une fois pour toute, à l'avance, la catégorisation de tout l'index d'un moteur de recherche, ou d'établir ce regroupement en fonction de la requête. Ce dernier traitement exige bien entendu plus de temps de calcul que le précédent. Les moteurs de recherche classiques comme *Google* ou *Yahoo*, ayant pour priorité de répondre d'une manière extrêmement rapide à la requête d'un utilisateur, ont bien entendu fait le choix de pré-traiter cette opération. Un tel système de classement *à l'avance* a notamment été décrit dans [Cutting *et al.*, 1992]. Dans ce travail, chaque document est représenté de manière vectorielle suivant l'ensemble de mots le composant et la mesure de similarité *cosine* est employée pour comparer les textes du corpus. Le volume de données à traiter étant extrêmement important, l'emploi de techniques de catégorisation classique ne peut pas être effectué. Deux algorithmes sont alors définis dont notamment l'algorithme

Buckshot qui se décompose en trois phases. La première consiste à établir des groupes initiaux par une technique de classification classique - comme la CAH - appliquée sur une petite portion de documents du corpus choisis aléatoirement. Le reste des documents est alors assigné à la catégorie la plus proche en fonction d'une distance définie par les mesures de similarité opérées sur les textes et le centre de chaque classe déterminée à l'étape précédente. Enfin, la dernière opération consiste à opérer un raffinement sur les catégories obtenues en réaffectant l'ensemble des documents obtenus à la catégorie dont le centre est le plus proche. Chaque groupe obtenu est alors labellisé par le mot le plus représentatif de la catégorie, au sens de sa fréquence d'apparition dans l'ensemble des documents de la catégorie.

[Hearst et Pedersen, 1996] effectue une comparaison entre une catégorisation pré ou post requête. Les résultats obtenus dans les analyses de comportements d'utilisateurs face à ces deux modes de calcul ont clairement montré qu'un post traitement réalisé sur les documents retournés par le système de recherche d'information permet de retrouver plus efficacement les *bonnes* réponses. En effet, en ne prenant en compte que les documents résultats pour établir les catégories, on élimine ceux ne correspondant pas à la requête qui introduiraient un biais dans la classification. Les groupes déterminés sont alors plus proches de ce que recherche l'utilisateur, facilitant grandement son discernement.

Un des plus important systèmes de post traitement mis en place de nos jours reste sans doute le système *Grouper* [Zamir et Etzioni, 1999]. En préambule à ce travail, Zamir définit dans [Zamir et Etzioni, 1998] trois éléments nécessaires à la réalisation de toute catégorisation intervenant sur les résultats d'une requête : 1) L'algorithme doit produire des groupes cohérents qui peuvent s'entrecroiser si nécessaire, 2) Des descriptions précises de chaque catégorie permettent à l'utilisateur de naviguer facilement dans les groupes et 3) L'affichage des résultats ne doit pas être significativement ralenti par le système de catégorisation.

Grouper est une interface de catégorisation du méta moteur *HuskySearch* ayant pour but de retourner les résultats obtenus par plusieurs moteurs de recherche à base d'index [Selberg et Etzioni, 1995]. *Grouper* regroupe les documents retournés par le méta moteur en fonction des phrases ou expressions qu'ils contiennent. Chaque groupe ainsi créé est considéré comme une catégorie de base. Les groupes possédant sensiblement les mêmes documents sont alors fusionnés afin de produire les catégories finales présentées à l'utilisateur. La figure 1.15 présente un aperçu des résultats obtenus. Les résultats sont présentés en fonction de leurs groupes d'appartenance en indiquant les expressions prises en compte dans chaque catégorie (en gras sur l'image). Seules les pages les plus représentatives de chaque groupe sont affichées par défaut afin de ne pas surcharger d'informations l'utilisateur mais tous les documents d'une catégorie sont accessibles en suivant les liens *View Results* de chaque groupe.

Toutes les techniques présentées ici concernent la classification non supervisée où les catégories sont déterminées de manière entièrement automatique. Les méthodes supervisées tiennent cependant une grande place dans les systèmes de recherche à base de catégories. Le plus connu d'entre eux est certainement *Yahoo* [Yahoo] dans lequel

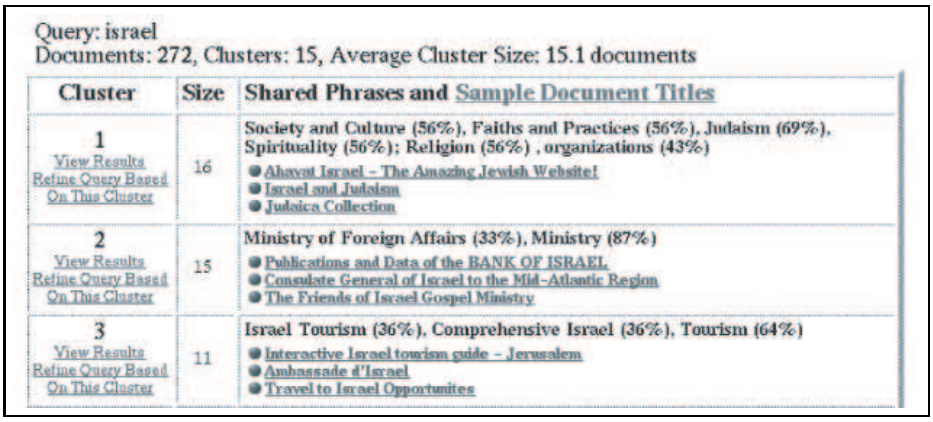


FIG. 1.15 – Résultats obtenus par *Grouper* sur la requête «*israel*».

les catégories et les documents étaient initialement sélectionnés de manière entièrement manuelle. Mais devant la vitesse à laquelle les documents apparaissent et disparaissent sur Internet, des méthodes de mises à jour automatiques ont été étudiées.

Les systèmes de classification supervisée ne peuvent exister qu'en pré-traitement d'une requête. [Chekuri *et al.*, 1996] montre l'exemple d'un système semi-automatique de construction de hiérarchie et d'interrogation d'un système de recherche associant mots-clés et catégories. Chaque document indexé par le système est associé à une catégorie. Le processus de classification est basé sur une analyse statistique des fréquences d'apparitions de termes. À partir d'une taxonomie pré-définie et de documents définissant des groupes, des vecteurs caractéristiques de fréquence de termes sont élaborés pour chaque catégorie. L'ajout de documents dans cette hiérarchie se fait alors de façon classique en fonction de la distance - mesure de similarité *cosine* - aux différents groupes définis. L'intérêt de ce système réside dans son interface d'interrogation. Celle-ci demande à l'utilisateur de fournir une liste de mots-clés et de sélectionner les catégories correspondant à la requête. Le système sélectionne alors classiquement par un index inversé les documents correspondant aux différents mots-clés demandés et filtre les pages retournées à l'utilisateur en fonction des catégories requises.

Plus récemment, des techniques d'apprentissage plus classiques de la littérature ont été utilisées. [Mladenic, 1998] utilise des classifieurs bayésiens sur des vecteurs représentatifs du texte des différents documents. Les résultats obtenus avec cette méthode sur la taxonomie de *Yahoo* montrent qu'environ 50% des éléments de test sont attribués dans des catégories correctes.

Chakrabarti aborde le problème d'apprentissage dans [Chakrabarti *et al.*, 1998b] d'une manière sensiblement identique à celle utilisée par Mladenic. La représentation des documents repose ici sur le modèle de Bernoulli. Chaque page est associée aux termes t la composant et à une catégorie c. Un classifieur bayésien est alors également utilisé sur les paramètres t et c. Lors de l'ajout d'un nouveau document, le choix du

groupe se fait en fonction de la classe maximisant la probabilité a posteriori d'appartenance à la catégorie basée sur le modèle de Bernoulli.

1.4.2.3 Présentation efficace de résultats sous forme de texte

L'affichage de résultats sous forme de texte n'offre pas beaucoup d'opportunités au concepteur afin de permettre à l'utilisateur une lecture aisée de la quantité importante d'informations mise à disposition. La structuration en catégories permet cependant quelques possibilités. Les impacts de telles représentations dans les moteurs de recherche ont été étudiés pratiquement depuis l'apparition de ces outils. Dès 1994, Chimera compara trois types de visualisations dans [Chimera et Shneiderman, 1994]. La première correspond à une liste complète de catégories et sous-catégories dans laquelle l'utilisateur navigue grâce à une barre de défilement. La deuxième, illustrée dans la figure 1.16 *a)*, permet de réduire les sous-catégories de manière à ne plus les afficher - dans l'exemple, seules les sous-catégories de la deuxième section sont développées. Enfin, la figure 1.16 *b)* montre l'utilisation de la technique des panneaux multiples où chaque panneau est dédié à un niveau de profondeur de l'arbre et correspond à l'élément sélectionné dans le panneau de niveau supérieur.

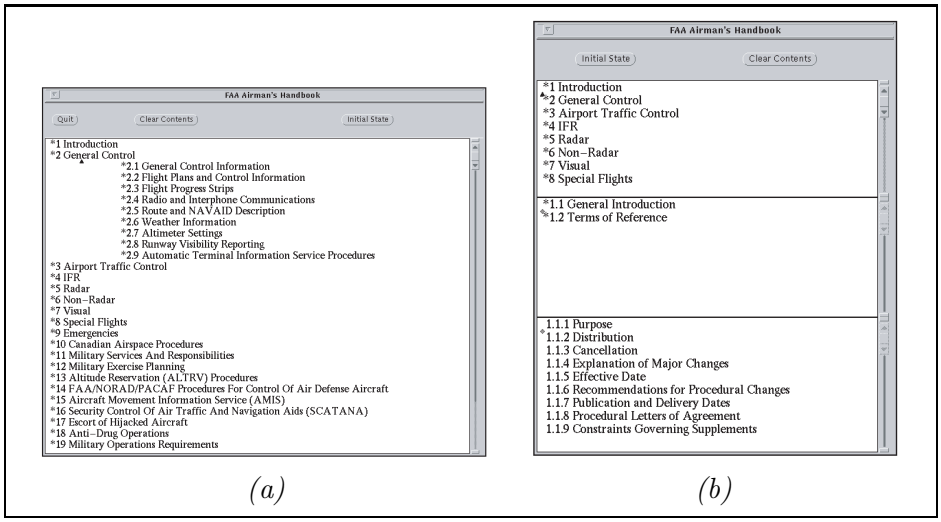


FIG. 1.16 – Affichage de listes de catégories *a)* à éléments développables *b)* à panneaux multiples.

L'étude a montré que la représentation de la totalité de la hiérarchie est très perturbante pour l'utilisateur et que la vue à panneaux multiples offre un confort de visualisation permettant de retrouver très rapidement l'information recherchée.

1.4.3 Visualisation graphique de résultats

La représentation de données de systèmes d'information sous forme graphique présente l'avantage de faire passer beaucoup d'informations très rapidement à l'utilisateur. Ces représentations peuvent intervenir en deux, trois et même quatre dimensions en prenant en considération la notion de temps comme nous le verrons dans la suite de cette section. Les travaux dans ce domaine sont nombreux et variés et semblent avoir un réel avenir prometteur dans le domaine de la recherche d'information.

Les études que nous décrivons dans les sections suivantes vont nous permettre d'établir les points importants à ne pas négliger dans la représentation des résultats issus de notre outil de recherche. Nous concevrons alors dans le chapitre 8 une méthode originale de visualisation de résultats.

1.4.3.1 Représentation en deux dimensions

Les systèmes de classement en catégories (voir section 1.4.2.2) trouvent ici un moyen efficace de représentation. [Carey *et al.*, 2000] introduit par exemple plusieurs systèmes de représentation de groupes de pages résultats d'une requête. La classification mise en œuvre utilise une adaptation de l'algorithme *Buckshot* [Cutting *et al.*, 1992] vu dans la section précédente. La variation introduite ici intervient dans la première phase : au lieu de choisir aléatoirement les pages prises en considération dans la CAH, on utilise les 150 premiers résultats obtenus. Le reste de l'algorithme est inchangé et caractérise les classes obtenues par des vecteurs de fréquence de mots.

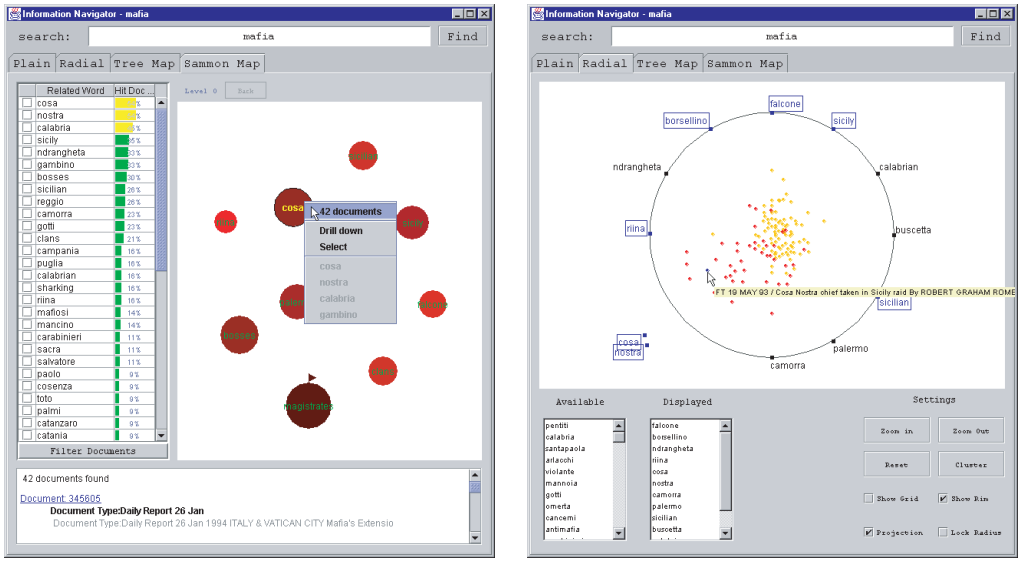
Afin de représenter les éléments multidimensionnels obtenus dans un espace à deux dimensions pour une lecture plus aisée des résultats, les auteurs utilisent les cartes de Sammon [Sammon, 1969]. L'algorithme décrit par Sammon établit la représentation souhaitée en tentant de conserver les distances entre les objets deux à deux. On effectue pour cela une recherche itérative de gradient. Les relations entre les vecteurs de catégories sont alors conservées.

La figure 1.17 *a)* montre l'interface ainsi obtenue. La représentation des résultats s'effectue à l'aide de trois panneaux servant respectivement à l'affichage des mots-clés les plus fréquents apparaissant dans les documents résultats, à la représentation de la carte des groupes déterminés et à l'affichage de détails de certains documents.

La carte fait apparaître les différents groupes sous forme de cercles identifiés par le mot le plus fréquent observé dans chacun d'eux. Le nombre de documents de chaque catégorie est indiqué à la fois par la couleur et la taille du disque la représentant, et la distance entre chaque cercle correspond à la similarité calculée entre chaque groupe de pages.

Les symboles utilisés ici correspondent simplement à ceux utilisés dans l'élaboration de cartes de densité de population. D'une manière générale, tous les systèmes de visualisation cherchent à utiliser des métaphores de représentation de leurs données connues de la plus grande majorité des utilisateurs afin de limiter leur temps d'apprentissage et de rendre la recherche plus efficace.

En sélectionnant une catégorie particulière, le détail des documents est représenté sous forme d'une liste classique d'éléments comprenant le titre et le résumé du texte.



(a) *(b)*

FIG. 1.17 – Représentation de catégories de documents en 2 dimensions *a)* par carte de Sammon *b)* et par visualisation radiale.

Le panneau contenant les mots-clés permet quant à lui de filtrer les pages en fonction des éléments désirés, ceci permettant à l'utilisateur de spécialiser sa requête en fonction des résultats obtenus.

Carey présente également dans le même article une autre visualisation dite *radiale interactive* illustrée par la figure 1.17 *b)*. Cette représentation est inspirée de plusieurs travaux, notamment [Hemmje *et al.*, 1994], [Hoffman *et al.*, 1999] et [Korfhage, 1991]. L'utilisateur agit ici de manière beaucoup plus interactive en sélectionnant les mots-clés représentatifs des pages qu'il désire afficher et en les positionnant autour du cercle défini sur le panneau de l'application. Les résultats du moteur de recherche sont représentés par des points attirés préférentiellement vers les mots-clés apparaissant dans leur texte avec une fréquence élevée. Un équilibre s'établit alors, figeant les documents à l'endroit où les forces d'attraction s'annulent. En déplaçant et choisissant d'autres mots-clés, l'utilisateur peut ainsi partitionner lui-même les résultats.

Le système *SQWID* initié par McCrikard dans [McCrickard et Kehoe, 1997] utilise le même type de visualisation mais propose une plus grande interaction avec l'utilisateur. La figure 1.18 montre l'interface de ce système. Le cœur de l'application est composé d'un modèle de tension afin de déterminer la position relative de documents à des termes sélectionnés par l'utilisateur. Le principal apport de *SQUID* réside dans le déplacement dynamique des pages qui se positionnent progressivement à leur place, laissant à l'utilisateur le loisir d'intervenir sur la disposition obtenue en manipulant chaque objet.

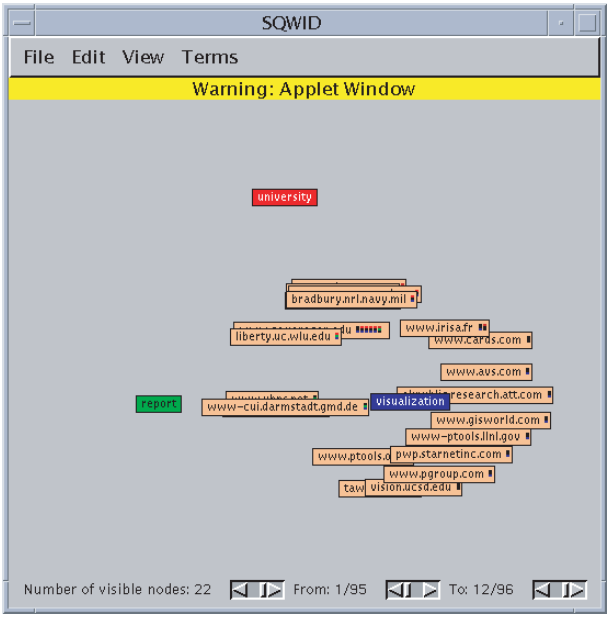


FIG. 1.18 – Interface de présentation des résultats du système *SQWID*.

SQWID agit comme un méta-moteur de recherche en interrogeant *Altavista* afin de récupérer les résultats d'une requête. Il détermine ensuite les termes qui caractérisent le mieux les résultats retournés en analysant le texte contenu dans chaque titre. Un classement des documents est alors établi pour chaque terme retenu en utilisant une fonctionnalité offerte par le moteur de recherche *Altavista*. Ce classement détermine la distance idéale séparant un couple terme-document. L'utilisateur peut alors sélectionner les mots-clés à prendre en compte dans le calcul des tensions et les placer à sa guise dans l'espace offert par l'interface graphique.

À l'initialisation, tous les documents sont centrés par rapport aux termes choisis et leur déplacement est calculé par la technique du recuit simulé. Cette méthode fait évoluer progressivement la position de chaque page en équilibrant les contraintes imposées par la position des termes dans l'interface.

Le système permet également de regrouper toutes les pages appartenant à un même site Web dans un nœud unique. Pour chaque nœud, des barres verticales représentent alors les documents regroupés. Une couleur est affectée à chaque terme sélectionné par l'utilisateur et est représentée sur chaque barre verticale avec une intensité proportionnelle à la similarité de la page au terme.

Ces interfaces ont montré que ce type de visualisation dynamique apporte un grand intérêt pour l'utilisateur qui peut extraire lui-même de l'information en toute simplicité.

1.4.3.2 Représentation hiérarchique

Le système des *tree-maps* est souvent utilisé pour représenter des données hiérarchiques. Les *tree-maps* ont initialement été proposées par Johnson et Shneiderman

dans [Johnson et Shneiderman, 1991, Shneiderman, 1992] pour représenter l'arborescence d'un système de fichiers. Ce type de visualisation a été conçu afin de représenter de manière humainement compréhensible des structures d'arbres complexes dans un espace à deux dimensions.

Les résultats d'un moteur de recherche pouvant prendre une forme hiérarchique, plusieurs travaux ont été réalisés afin d'adapter ce modèle de représentation aux systèmes de recherche d'information. Nous citerons notamment le système de Carey vu précédemment [Carey *et al.*, 2000] et les travaux de Shneiderman dans ce domaine [Shneiderman *et al.*, 2000]. La figure 1.19 illustre des résultats de recherche affichés sous forme de *tree-map*.

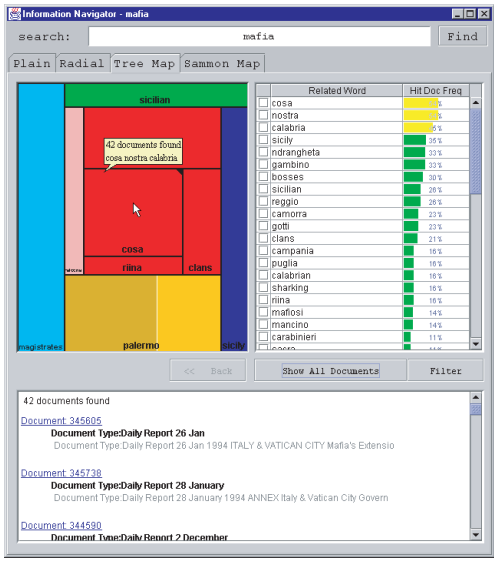


FIG. 1.19 – Représentation d'une hiérarchie de résultats par une tree-map.

Dans une *tree-map*, les catégories de documents sont représentées par des rectangles. L'ensemble de ces éléments est arrangé de manière à être inscrit totalement dans un plus grand rectangle. Les rectangles représentant les documents sont également conçus afin que leur taille soit proportionnelle au nombre de documents formant chaque groupe. Le principe de cette visualisation est de regrouper entre eux les éléments les plus similaires dans des sortes de *super groupes*.

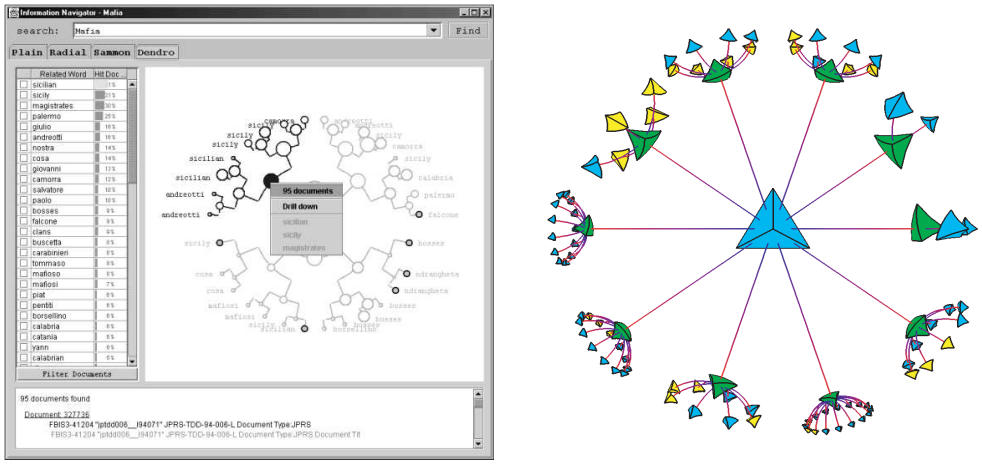
La délimitation entre deux *super groupes* est marquée de manière plus forte à l'aide de traits de séparation plus épais. De plus, le jeu de couleur utilisé renforce cette similarité, chaque *super groupe* étant représenté par des variations de teinte sur une même couleur de base. Ceci ajoute de la cohérence visuelle à la représentation globale. Afin de ne pas rajouter de la complexité pour la lecture de ce graphe, les catégories sont uniquement repérées par le mot-clé le plus fréquent apparaissant dans l'ensemble des documents de cette dernière. L'utilisateur peut évidemment accéder à plus d'information - nombre de documents, textes des documents, mots-clés les plus représentatifs - en cliquant sur un groupe.

Lorsque le nombre d'éléments à afficher dans la *tree-map* devient trop important, il est possible d'effectuer une sorte de zoom sur une catégorie particulière. L'affichage est alors remis à jour en ne prenant en compte que le *super groupe* sélectionné. On peut faire ici l'analogie avec une représentation fractale où il existe plusieurs niveaux de lecture, de la vue générale à la plus particulière. C'est l'intérêt principal de cette représentation.

Il y a bien évidemment d'autres manières de représenter une hiérarchie. Mais les données hiérarchiques elles-même peuvent être d'une toute autre nature. La structure d'un site apportant en effet une information significative, il peut être intéressant de la visualiser dans l'optique d'un affichage de résultats d'une recherche d'information.

Cette structure prend la forme d'arbres et peut également être représentée par un dendogramme. Cependant, l'arbre engendré étant de taille généralement conséquente, la visualisation est déformée à l'aide de sortes de loupes, comme par exemple à l'aide d'un affichage hyperbolique. La déformation permet à l'ensemble de l'arbre d'être visualisé dans l'espace de représentation en réservant une grande partie de l'affichage à une petite partie de l'arbre et en repoussant le reste des éléments à afficher aux limites de l'espace disponible.

Carey a également ajouté ce type de visualisation à son système *Information Navigator* dans [Carey et Heesch, 2003] représenté sur la figure 1.20 *a)*. L'arbre peut ainsi être visualisé à partir de n'importe quel point, le reste du graphe étant étalé sur l'ensemble de l'espace de représentation. L'effet de loupe sur le centre de la visualisation permet de regarder en détail les parties que l'on juge intéressantes.



(a) *(b)*

FIG. 1.20 – Affichage d'arbre en vue hyperbolique *a)* de classifications de résultats *b)* et de représentation de sites web.

Webviz, décrit dans [Munzner et Burchard, 1995], est un système de visualisation de portions d'Internet utilisant une représentation hyperbolique. Sa conception graphique se rapproche d'un affichage 3D que nous aborderons dans la section suivante. Ce système permet de montrer l'organisation des documents pris en compte en décrivant les liens

les reliant. Ce travail trouve naturellement un écho important dans la visualisation de l'architecture de sites Web.

Webviz utilise une représentation arborescente des navigations possibles au sein d'un site. Cependant, le Web est un graphe, il est donc nécessaire d'adopter une stratégie particulière pour construire cet arbre. Ce dernier est généré en considérant une page de départ comme nœud initial. Les fils de ce nœud sont représentés par les pages cibles des liens hypertextes du document et ainsi de suite pour les nœuds inférieurs. La figure 1.20 *b)* montre un exemple de l'interface de *webviz* visualisant un site dans lequel les liens retours vers des pages déjà visitées ont étés supprimés. Il est cependant possible de représenter non pas un arbre mais un graphe en indiquant tous les liens existant entre toutes les pages. Mais en pratique, cela a tendance à surcharger l'affichage et à perdre l'utilisateur dans son cheminement dans le graphe.

1.4.3.3 Représentation en trois dimensions

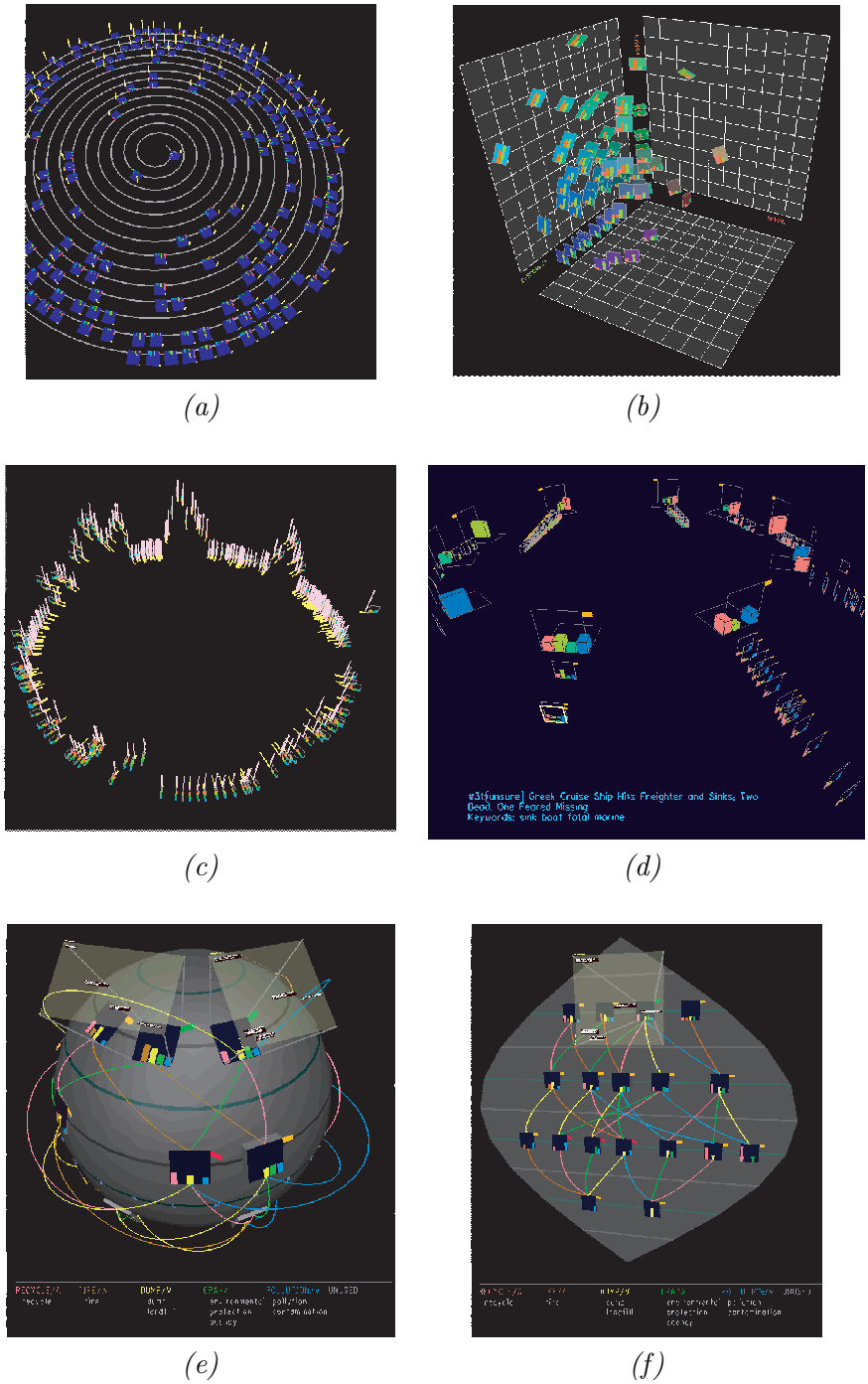
Des systèmes plus élaborés proposant des interfaces en trois dimensions ont également vu le jour. L'ajout d'une dimension supplémentaire permet de capitaliser encore un peu plus le nombre d'informations affichées tout en ne contraignant pas l'utilisateur à un apprentissage fastidieux d'une visualisation de données.

Dans cette optique, Cugini a expérimenté différents types d'interfaces au travers de plusieurs travaux consécutifs [Cugini *et al.*, 1996, Cugini *et al.*, 2000]. Son étude est intéressante parce qu'elle aborde un grand nombre de métaphores de représentation graphique. De plus, chaque élément a été testé par des utilisateurs réels dans le cadre de l'élaboration du prototype *NIRVE* (NIST Information Retrieval Visualisation Engine). Ces interfaces utilisent les données retournées par le moteur de recherche *PRISE* développé par la même équipe. À partir d'une requête de mots-clés, ce moteur renvoie une suite d'informations sur chaque document : un identifiant unique, un titre, un score relatif à sa pertinence à la requête, un rang, une taille et le nombre d'apparitions de chaque mot-clé dans le texte.

Le principe de base de chaque type de visualisation est de présenter en priorité une vue globale de la structure des résultats plutôt que de se concentrer sur la recherche d'un document particulier. Dans tous les modèles, l'utilisateur peut se déplacer autour de la représentation 3D et sélectionner les icônes représentant les documents afin de visualiser des informations plus précises comme par exemple le texte ou des statistiques sur la présence des mots-clés.

Le premier modèle testé tente de préserver l'information de rang retourné par le moteur de recherche en représentant les documents ordonnés au sein d'une spirale (voir figure 1.21 *a)*). Le centre de la spirale contient les pages correspondant le mieux à la requête et l'espacement entre deux icônes est proportionnel au score de chaque page.

L'utilisateur peut naviguer au sein de cette spirale en partant par exemple du milieu et en s'éloignant petit à petit vers la périphérie. Chaque page est représentée par un cube sur lequel sont affichées les fréquences d'apparition des mots-clés grâce à des barres de couleurs. Cette interface offre également la possibilité d'attribuer différents poids d'importance à chaque mot de la requête. Lorsque une modification de poids à lieu, les



(a) *(b)* *(c)* *(d)* *(e)* *(f)*

FIG. 1.21 – Présentation de résultats en 3-D de *NIRVE* *a)* sous forme de spirale, *b)* sous forme de graphe, *c)* regroupés en catégories en utilisant la méthode du plus proche voisin, *d)* regroupés en catégories en alignant les documents similaires, *e)* en utilisant une métaphore satellitaire et *f)* en utilisant une métaphore cartographique.

cubes représentant les documents sont élevés au-dessus de la spirale proportionnellement au produit de la fréquence d'apparition du mot-clé concerné et de son poids. L'utilisation de la troisième dimension offre ici un classement alternatif basé sur l'importance que l'utilisateur donne aux mots de la requête.

Cependant, cette représentation n'est pas exempte de défauts. Les utilisateurs ont généralement du mal à comprendre qu'un groupe de documents apparaissant côte à côte sur l'image ne corresponde pas réellement à une catégorie de pages ayant des points communs. Ceci est dû au fait que quelle que soit la métaphore utilisée, les utilisateurs conçoivent une représentation graphique comme une représentation spatiale et interprètent toujours la distance euclidienne entre deux éléments comme une indication de proximité.

La deuxième interface réalisée (figure 1.21 *b)*) cherche à palier ce problème et représente les résultats sur un graphique classique à trois dimensions.

L'utilisateur choisit trois mots-clés de la requête qui vont être affectés aux trois axes de représentation. La fréquence d'apparition de ces mots-clés dans les documents établit les coordonnées de ces derniers. Les résultats sont représentés par des icônes indiquant le profil de fréquence - sous forme d'histogramme - de tous les mots-clés initiaux. Une extension de cette représentation autorise l'utilisateur à spécifier plusieurs mots par axe. Les coordonnées correspondent alors à la moyenne de toutes les composantes de l'axe.

Cette projection des résultats souffre de problèmes classiques de sélection d'axes de représentation. La plupart des documents se retrouvent localisés dans un espace généralement proche de l'origine des axes et il devient très difficile de les distinguer. Les auteurs n'ont par ailleurs pas réellement observé de création de groupes de documents se détachant en fonction des mots choisis. Ceci peut s'expliquer par le fait que les documents analysés et pris en compte dans cette représentation ont été sélectionnés par le moteur de recherche dans le but de maximiser la fréquence d'apparition des mots de la requête. Ces constatations montrent clairement qu'une classification des résultats ne peut être réalisée en tenant uniquement compte des mots de la requête. Il est également important de noter que dans une telle représentation, l'utilisateur doit pouvoir se repérer facilement dans l'espace grâce notamment à des plans fixes - ici les plans des axes principaux - sous peine de désagréments et de rejet de l'interface.

La figure 1.21 *c)* présente un autre type de visualisation dont le but est de simplifier l'affichage et d'apporter une analyse supplémentaire facilitant la lecture des résultats. Les auteurs définissent une mesure de similarité entre les documents afin de les classer en groupes. La mesure utilisée est simple et correspond à la distance euclidienne mesurée entre les profils de fréquence des mots-clés apparaissant dans les textes des documents, chaque profil représentant un vecteur de dimension égale au nombre de mots de la requête.

Les icônes représentant les documents sont disposées sur un cercle en plaçant côte à côte les éléments les plus similaires entre eux. De la même manière que pour la méthode d'affichage en spirale représentée figure 1.21 *a)*, la troisième dimension est utilisée. Elle symbolise ici la distance - au sens de la mesure de similarité - entre deux éléments

consécutifs. Ainsi, plus un document est élevé, plus il diffère de son successeur, la lecture se faisant dans le sens inverse des aiguilles d'une montre. Les trous apparaissant dans la représentation agissent comme des séparateurs de classes de documents.

On constate que la structure circulaire ne pose pas le problème de jugement spatial rencontré dans le modèle à spirale. Cependant, l'utilisation de nombreux mots-clés dans la requête - en nombre supérieur à 5 - entraîne des difficultés de visualisation, notamment au niveau de la coupure des groupes, et rend la compréhension du schéma généré beaucoup plus difficile.

L'interface de visualisation suivante (figure 1.21 *d)*) essaye de gommer quelques erreurs déterminées dans l'analyse des résultats obtenus précédemment. Le regroupement par profil de fréquence de mots-clés entraîne une trop grande différenciation de mots similaires, comme par exemple *tornado* et *twister*. Par ailleurs, lorsque un utilisateur cherche à spécialiser fortement une requête, il emploie des mots très similaires entre eux. Ceci handicape évidemment beaucoup la représentation de groupes précédente. Un pré-traitement des mots-clés est alors nécessaire et l'utilisateur a la possibilité de regrouper plusieurs mots en concepts de manière à créer des catégories les plus différentes entre elles. Ce changement a de plus l'avantage de réduire la dimension des vecteurs de profils des documents et de simplifier la visualisation de ceux-ci.

Le regroupement de documents est réalisé de la même manière en rapprochant de proche en proche les documents les plus similaires entre eux. Les groupes sont établis en fonction d'un seuil de différence de similarité et sont représentés graphiquement de manière beaucoup plus explicite comme on peut le voir sur la figure 1.21 *d)*. Ce paramètre est modifiable en permanence par l'utilisateur et les changements sont affectés dynamiquement sur la représentation graphique des résultats. Les icônes alignées sur un cercle représentent le profil moyen des pages appartenant à un groupe. L'angle les écartant deux à deux est proportionnel à la distance - en terme de similarité - les séparant. Le profil de chaque page est ensuite affiché derrière le profil du groupe. La distance séparant ces deux derniers profils est calculée de manière à être proportionnelle à sa similarité au groupe.

Il est également possible dans cette visualisation d'attribuer des poids d'importance relatifs à chaque concept de manière à faire évoluer la classification pour mieux cerner les résultats. Mais force est de constater que peu d'utilisateurs utilisent ce principe jugé trop subtil.

L'expérimentation a montré que les utilisateurs ont tendance à modifier le seuil proposé afin de générer des groupes dont les profils contiennent le même nombre d'éléments non nuls. L'information recherchée correspond alors à l'absence ou à la présence d'un concept dans les documents. La visualisation présentée dans la figure 1.21 *e)* a pour but de caractériser les groupes de documents dans ce sens. L'algorithme de classification est ici simplifié et regroupe les pages ayant des mots-clés appartenant aux mêmes concepts.

La métaphore de visualisation utilisée ici est représentée sur deux dimensions - cercle et spirale n'en possédant qu'une - et ressemble à une carte de satellites répartis autour du globe terrestre. La latitude des éléments indique le nombre de concepts présents : le

pôle nord contient les éléments possédant tous les concepts, le pôle sud les éléments dans lesquels ne figurent aucun concept. Sur une même latitude, la longitude sépare alors les profils différents, autrement dit les groupes possédant le même nombre de concepts mais répartis différemment. Le nombre de documents par groupe est symbolisé par l'épaisseur de la boîte représentant le groupe. Enfin, des liens sont disposés entre les groupes pour lesquels le profil ne diffère que d'un seul élément.

Les tests réalisés montrent que la visualisation perd de son intérêt si le nombre de concepts est trop important, typiquement plus de quatre ou cinq, en rendant l'affichage trop complexe. La représentation 3D pose ici quelques difficultés d'adaptation pour les utilisateurs novices. La navigation 3D est une question d'apprentissage et d'habitude, les experts utilisant l'informatique intensivement retrouvent assez rapidement l'information dans cette métaphore visuelle satellitaire.

Enfin, le modèle présenté dans la figure 1.21 *f)* tente de capitaliser toutes les informations récoltées par l'utilisation des interfaces précédentes. La représentation utilise le même concept de regroupement que l'exemple précédent mais disposé sur un plan à deux dimensions. Les groupes sont ainsi alignés sur plusieurs lignes, chacune correspondant à un nombre de concepts présents dans le profil. Les liens déterminés entre les catégories sont conservés également et affichés sous forme d'arcs en utilisant la troisième dimension.

Afin de pouvoir examiner le contenu de chaque groupe, l'utilisateur sélectionne une icône particulière. Les documents le composant sont projetés sur un plan parallèle au plan de visualisation principal comme on peut le voir sur la figure 1.21 *f)*.

Cette visualisation simplifiée par rapport aux précédentes convient mieux aux utilisateurs. Elle requiert moins d'apprentissage et s'appréhende plus rapidement qu'une visualisation 3D. L'ajout de quelques petites entités en trois dimensions intégrées dans un environnement à deux dimensions facilite cependant l'attrait et les possibilités offertes par un système de cette nature.

La leçon à retenir des travaux de Cugini est que l'emploi d'un modèle de visualisation trop sophistiqué en 3D n'est pas recommandé pour les utilisateurs non avertis. Cependant, lorsque la visualisation emprunte une métaphore visuelle bien connue des utilisateurs, la conclusion est toute autre. Les travaux réalisés par Mukherjea en la matière en sont un parfait exemple [Mukherjea et Foley, 1995, Mukherjea et Hara, 1999].

Selon leurs principes, une bonne visualisation doit permettre à la fois d'afficher une information globale et une information précise sur l'environnement. En cela, l'affichage hyperbolique représente un bon compromis, mais il existe d'autres moyens de réaliser cette tâche à l'aide de métaphores bibliothécaires par exemple (voir figure 1.22).

1.4.3.4 Visualisation à l'aide de métaphores

Un ensemble de cartes représentant chacune une page résultat d'une requête est proposé à l'utilisateur. La profondeur de rangement des cartes est ici représentative de la pertinence des résultats proposés. Chaque carte représente en son centre une page

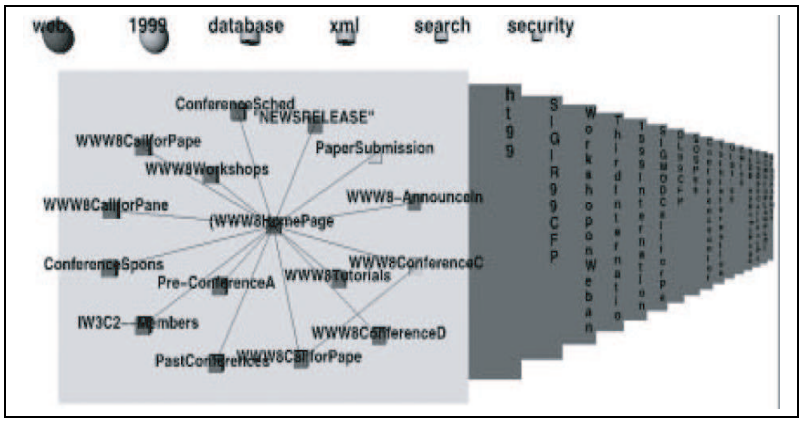


FIG. 1.22 – Représentation d'une hiérarchie de résultats en 3-D par métaphore bibliothécaire.

retournée par le moteur de recherche mais propose également le graphe de liaison de cette page à ses plus proches voisins. Cela permet à l'utilisateur de déterminer le contexte du site dans lequel apparaît un résultat et de cerner plus rapidement la pertinence de ce dernier.

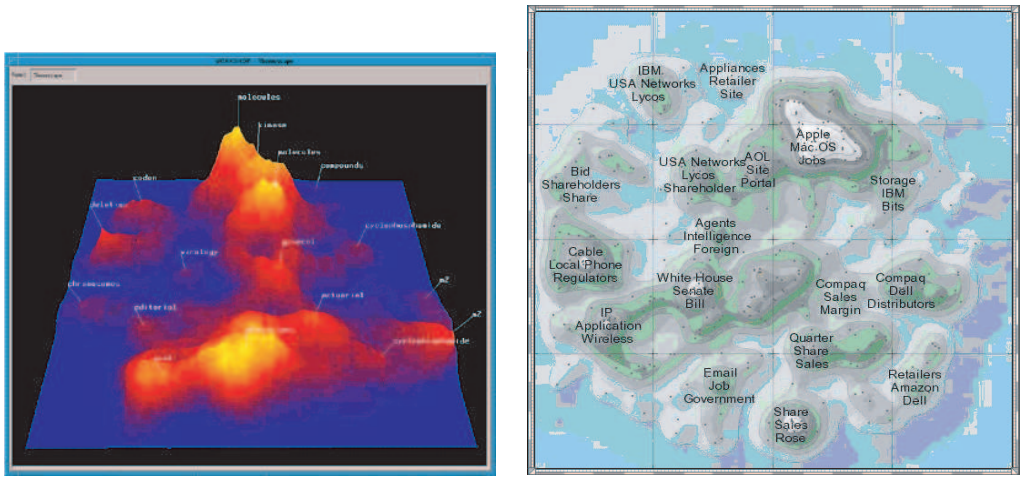
Chaque carte est nommée par le titre du nœud principal de son graphe. De plus, la partie supérieure de la visualisation permet d'afficher les termes de la requête sous forme de cylindres ainsi que les mots les plus fréquents retrouvés dans les résultats. En cliquant sur un des mots présentés, les nœuds du graphe actuellement affichés ainsi que les cartes correspondant au mieux au terme sélectionné sont soulignés.

Des méthodes similaires existent comme par exemple *LibView* représentant une bibliothèque de documents que l'on peut sélectionner individuellement afin de visualiser de manière plus précise leur organisation interne en fonction des termes d'une requête [Rauber et Bina, 1999]. On peut signaler également la technique des *murs en perspective* qui affiche sur un mur en trois dimensions les résultats d'une requête [Mukherjea et Foley, 1995, Mackinlay *et al.*, 1991]. Dans cette visualisation, les catégories de documents sont séparées par les différents pans du mur. Chacune de ces représentations utilise à la fois une vue générale de l'ensemble des documents et une vue plus détaillée d'une partie de ces documents.

Les représentations de résultats à base de métaphores cartographiques sont également de très bons exemples de visualisations appréciées et comprises très rapidement par une grande majorité des utilisateurs. Les travaux sur ce type de représentation ont commencé dès le début de la généralisation des moteurs de recherche sur Internet. Chalmers représente ainsi dès 1993 les résultats d'une requête sous forme de vallées et de collines dans [Chalmers, 1993]. Les documents sont représentés sur une surface rectangulaire en fonction de leur similarité à certains mots-clés disposés sur la carte. Lorsque la densité des documents est importante, une colline émerge tandis qu'une faible densité

de documents fait apparaître une sorte de vallée.

Dans le système SPIRE [Wise *et al.*, 1995, Wise, 1999], la signification des montagnes représentées sur la carte est différente. Les documents sont disposés sur le plan de la représentation par projection des résultats obtenus par la classification de l'ensemble des documents. Cette classification est réalisée par l'algorithme *K-Means* en utilisant une représentation vectorielle des documents et la mesure de similarité *cosine* (voir section 1.1.3). La hauteur d'affichage de chaque document est alors calculée en fonction de leur pertinence aux mots-clés. Un lissage est enfin opéré sur l'image pour s'approcher un peu plus de la métaphore. La figure 1.23 *a)* illustre cette représentation. La projection des classes de documents sur le plan est réalisée de manière à éloigner des groupes dissemblables. Ainsi, deux collines éloignées dans le sens euclidien du terme correspondent à deux groupes de documents dissemblables.



(a) *(b)*

FIG. 1.23 – Affichage de résultats du système SPIRE sous forme de carte terrestre *a)* en 3D, *b)* en 2D.

Cette visualisation permet de faire ressortir naturellement les groupes de documents correspondant le mieux à la requête initiale. Cependant, la visualisation en 3D n'est pas obligatoire et a tendance à surcharger l'affichage. En partant du constat que tout le monde a l'habitude de lire des cartes topographiques représentées en deux dimensions grâce à un code de couleur, les auteurs ont simplifié le système en utilisant ces repères communs à tous les utilisateurs. La figure 1.23 *b)* montre les résultats obtenus.

La majorité des systèmes de visualisation présentés ici tendent à illustrer le résultat du regroupement de documents en catégories. Mais la manière même dont ces documents se regroupent est porteur d'une information qui peut aider l'utilisateur à mieux interpréter les groupes qui lui sont présentés. La dimension temporelle permet d'apporter une réponse adéquate à ce problème.

1.4.3.5 Représentation en quatre dimensions

Il peut donc être intéressant de montrer à l'utilisateur la formation des classes dans un algorithme de classification. Cela lui permet de mieux comprendre les résultats obtenus et ainsi d'être plus efficace dans leur traitement. Les méthodes de classification à base de nuages d'insectes sont de très bons candidats à ce type de représentation.

De tels algorithmes sont présentés dans [Proctor et Winter, 1998]. Ils s'inspirent des insectes volants ou des bancs de poissons se déplaçant en nuage (voir figure 1.24). Ces derniers créent des mouvements complexes à partir de règles locales simples. Chaque individu représente une donnée et leurs déplacements visent à créer des groupes de données homogènes se déplaçant ensemble dans un espace à trois dimensions. Les groupes sont créés et visualisés en temps réel. L'utilisateur peut alors comprendre plus facilement par exemple pourquoi et comment des données similaires se regroupent ou de quelle manière les données isolées représentent des cas à part.

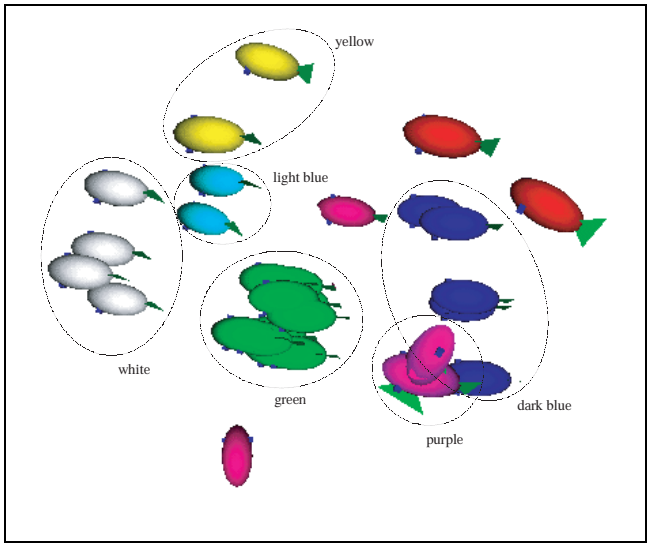


FIG. 1.24 – Classification par nuages d'agents. Cette modélisation s'inspire des insectes volants ou des bancs de poissons se déplaçant en nuage.

Nous reviendrons sur cette méthode dans le chapitre 8 dans lequel nous avons adapté ce type de visualisation au modèle de recherche que nous avons créé. Elle nous permet d'établir une classification des résultats fournis par notre méthode de recherche et de les présenter de manière visuelle facilitant la compréhension de leur classement.

1.4.4 Critique des différentes représentations

De nombreux auteurs ont tenté de cerner les éléments visuels permettant réellement à un utilisateur d'exploiter pleinement les résultats fournis par un moteur de recherche. On peut distinguer notamment les études menées par Nowell et Veerasamy sur

les visualisations à deux dimensions [Nowell *et al.*, 1996, Veerasamy et Belkin, 1996, Veerasamy et Heikes, 1997]. D'autres travaux concernent plus particulièrement les représentations en 3D et le rapport entre complexité d'affichage et niveau d'expertise des utilisateurs [Lamping *et al.*, 1995, Swan et Allan, 1998, Sebrechts *et al.*, 1999].

Cependant, toutes concluent que chaque méthode de représentation d'information nécessite un certain apprentissage de la part des utilisateurs du système. Un système graphiquement évolué, présentant une information condensée ou résumée de quelque manière que ce soit, doit être totalement comprise pour favoriser son adoption par les utilisateurs.

Quel que soit le système de représentation utilisé, les informations affichées doivent être sélectionnées afin de ne laisser paraître que l'essentiel. Lorsque trop d'informations sont disponibles dans une même représentation, l'utilisateur ne peut les intégrer toutes ce qui nuit à la compréhension de l'ensemble des résultats. Le regroupement de documents en catégories est généralement bien accepté et compris par les utilisateurs de système de RI et obtient toujours un accueil favorable quel que soit le niveau d'expertise du public concerné. La catégorisation a en effet l'avantage de compresser l'information afin de ne distinguer, à un haut niveau, que les éléments les plus dissemblables. Une vue hiérarchique permet également d'offrir plusieurs niveaux de détails guidant l'utilisateur d'une information très générale vers des sujets très précis au rythme de sa lecture des résultats.

Il n'existe par conséquent pas de représentation idéale de résultats pour un moteur de recherche, mais un concepteur de tels systèmes doit garder à l'esprit la simplification des informations à transmettre.

1.5 Conclusion

Comme nous l'avons vu, de nombreux travaux ont été menés sur la problématique de la recherche d'information. Il s'agit en effet d'un problème crucial dans une époque ou la quantité d'information accessible en permanence devient extrêmement importante. Il est en effet à l'heure actuelle impossible de se passer d'outils de recherche automatiques et de réaliser cette tâche de manière manuelle.

Les systèmes étudiés dans ce chapitre sont très divers et couvrent de nombreux domaines de l'informatique. Chacun établit une certaine avancée dans le processus global et permet à la fois de mieux détecter et trier les *bonnes* informations et de les présenter de manière intelligible.

Nous avons ainsi décrit les différentes formulations de requêtes utilisées actuellement dans les systèmes de recherche d'information, proposé un état de l'art des techniques et algorithmes utilisés dans les grands moteurs de recherche et présenté des méthodes d'affichage de résultats permettant à un utilisateur de les interpréter efficacement. Cet état de l'art nous permet de définir les grandes lignes de cette thèse et de les positionner par rapport à l'existant :

– les systèmes d'interrogations utilisés dans les moteurs de recherche actuels sont

basés principalement sur le modèle booléen. Ce modèle reste pauvre pour l'utilisateur qui voudrait, notamment dans le cadre de la veille stratégique, définir plus précisément le type de documents recherchés. Les systèmes interactifs présentant des résultats à l'utilisateur et lui demandant une évaluation sont une manière de dépasser les limites du modèle booléen, mais ils sollicitent trop l'utilisateur selon notre point de vue. Nous allons donc définir une formulation de la requête qui puisse dépasser le cadre booléen. Nous nous servirons du fait que notre approche analyse en détail le contenu des pages. L'utilisateur pourra ainsi spécifier des contraintes sur le contenu des documents recherchés.

– l'architecture des moteurs classiques est aujourd'hui caractérisée par l'utilisation d'un index inversé et d'un ensemble de machines fonctionnant en parallèle, à l'instar de Google. La pertinence des réponses est liée à un système de tri de pertinence construit sur la notion de lien existant entre les pages. Dans ce cadre, nous allons proposer des stratégies de recherche équilibrant de différentes manières l'effort de recherche entre l'exploitation des résultats fournis par les moteurs classiques et l'exploration de nouveaux liens dans les pages analysées. Nous pensons que ces nouvelles stratégies de recherche pourront accéder plus rapidement aux documents recherchés, en permettant de plus l'utilisation d'une requête plus riche comme mentionnée dans le point précédent. À partir des différentes lois dégagées de l'analyse de la topologie d'Internet, nous allons chercher à adapter une méthode de recherche similaire à celles existant dans le domaine de l'optimisation. Pour cela, nous allons construire une modélisation d'Internet, présentée dans le chapitre 3, établissant une nouvelle définition du problème de recherche d'information sur de vastes réseaux. Nous allons définir dans le même but deux opérateurs parcourant l'espace de recherche à la découverte de nouvelles pages. Enfin, nous allons utiliser une architecture réellement parallèle pour accélérer l'exécution de ces stratégies.
– les systèmes de visualisation actuels de résultats de moteur restent encore marginaux. Ils utilisent des techniques classiques de classification ou de visualisation d'information, comme l'utilisation de points d'intérêts dans le système Sqwid. Nous allons présenter une nouvelle approche pour établir dynamiquement une classification visuelle des résultats s'inspirant du comportement d'une population d'agents en déplacement.

Dans le chapitre suivant, nous détaillons trois types de méta-heuristiques que nous adapterons à notre problématique de RI. Il s'agit des algorithmes génétiques, des algorithmes à base de populations de fourmis artificielles et des algorithmes tabou. L'utilisation et l'adaptation de ces algorithmes pour la RI seront présentées respectivement dans les chapitres 4, 5 et 6. Nous élaborerons un système de classification visuelle dans le chapitre 8 simplifiant le processus d'analyse de l'utilisateur.

Chapitre 2

Méta-heuristiques et recherche d'information

Résumé

Ce chapitre aborde plusieurs méta-heuristiques trouvant principalement leurs sources d'inspiration dans le courant des algorithmes évolutionnaires, ainsi que leur adaptation à des problèmes de recherche d'information sur Internet. Ces méta-heuristiques vont nous servir par la suite de base à l'élaboration de plusieurs algorithmes (génétiques, inspirés du comportement des fourmis et basés sur un algorithme tabou) permettant de résoudre le problème posé dans cette thèse.

2.1 Introduction aux algorithmes génétiques

2.1.1 Principes généraux

Les algorithmes génétiques [Holland, 1975], et plus généralement les algorithmes évolutionnaires [Fogel *et al.*, 1966, Koza, 1992a, Schwefel et Rudolph, 1995], se basent sur les grands principes rencontrés dans la nature et notamment celui de l'évolution des espèces et de la sélection naturelle énoncés par Charles Darwin [Darwin, 1859].

Cette théorie explique comment, depuis l'apparition de la vie, les espèces ont su évoluer de générations en générations dans le sens d'une meilleure adaptation des individus à l'environnement. En favorisant la survie et la reproduction des individus bien adaptés à l'environnement, cette évolution a su amener des organismes unicellulaires à la vie telle que nous la connaissons aujourd'hui dans toute sa diversité.

La nature assure, de cette manière, la pérennité des meilleures caractéristiques de chaque individu. La recombinaison de ces caractéristiques entre deux individus permet de former au fil des générations de nouveaux individus pouvant être mieux adaptés à leur environnement ; chaque enfant reçoit en effet des caractéristiques à la fois de son père et de sa mère.

La génétique a su donner une explication au phénomène d'évolution grâce à la découverte de l'ADN en 1944 par Thomas Avery. Toutes les informations nécessaires à la genèse d'un individu (son patrimoine génétique) sont contenues dans les molécules d'ADN, composant les chromosomes, dont chaque cellule de l'organisme possède une copie. L'hérédité consiste à échanger des brins de chromosomes entre individus lors de la reproduction et ainsi à mélanger le patrimoine génétique de deux individus.

La mutation de gènes introduit des erreurs dans ce processus de copie. Elle peut contribuer à améliorer les individus et notamment de les adapter à un changement d'environnement.

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation s'appuyant sur les principes d'évolution naturelle et de sélection des espèces comprenant des *croisements* entre individus, des *mutations* apparaissant aléatoirement au sein d'une population et une *sélection* des individus les mieux adaptés à l'environnement [Holland, 1975]. Face à un problème pour lequel il existe un grand nombre de solutions, l'AG va explorer l'espace des solutions en se laissant guider par les principes décrits précédemment. On peut présenter les mécanismes mis en jeu dans les algorithmes évolutionnaires et plus particulièrement dans les AG par l'organigramme de la figure 2.1. [Venturini, 1997] apporte une synthèse des travaux réalisés sur les AG et leurs applications.

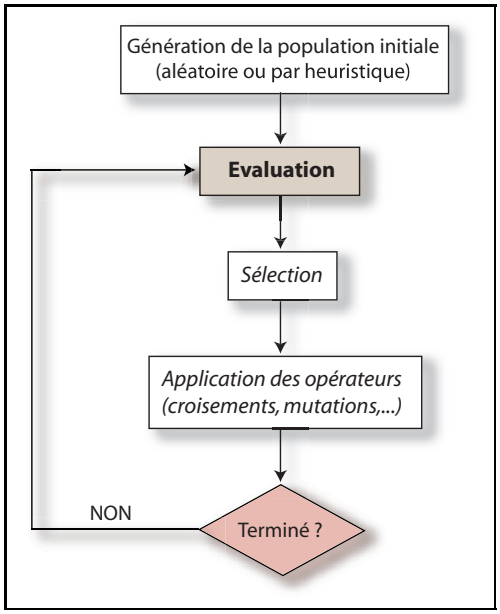


FIG. 2.1 – Organigramme d'un algorithme évolutionnaire.

Les AG ont déjà une histoire relativement ancienne [Holland, 1975]. Le vocabulaire employé dans son ouvrage est directement repris de celui de la théorie de l'évolution et de la génétique. Nous parlerons donc d'individus (solutions potentielles), de population, de gènes (variables), de chromosomes, de parents, de descendants, de reproduction, de croisement et de mutation.

Holland définit ainsi un *organisme* - l'individu - qui comprend plusieurs chromosomes eux-mêmes composés de gènes (morceaux continus d'ADN). Ces organismes composent la population que va faire évoluer l'AG suivant des lois nommées *reproductive plans*.

Il doit être également possible d'évaluer un individu sur la base de ses performances sur un problème donné. Cette évaluation doit permettre de mettre en œuvre une pression sélective efficace sur la population afin de la faire tendre vers un optimum.

Un AG se définit alors premièrement par le codage employé afin de caractériser les individus étudiés. Ce codage établit une convention qui va permettre de décrire chaque solution possible sous la forme d'une suite de caractères. Il est très important de prendre soin de la définition de ce codage car il conditionne fortement la complexité finale de la méthode. La population initiale constitue un deuxième paramètre. Holland préconise une initialisation aléatoire et uniforme mais il est possible d'adopter une heuristique dépendante du problème traité et permettant de faire converger plus rapidement l'AG vers une bonne solution. Enfin, il faut définir précisément une fonction d'évaluation - ou fonction de *fitness* - qui pour chaque solution possible donnera une valeur reflétant sa qualité pour résoudre le problème posé ainsi que des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche. L'opérateur de croisement recompose les gènes d'individus existant dans la population. L'opérateur de mutation a pour but de garantir l'exploration de l'espace de recherche.

À la fin des années 80, Goldberg a contribué à populariser les AG en définissant l'AG canonique où les individus sont décrits par l gènes binaires et sont évalués numériquement par une fonction d'évaluation f [Goldberg, 1989]. Il se décompose en trois étapes principales décrites ci-dessous.

1. **Générer** aléatoirement et uniformément une population initiale $P(0)$ de n individus,
   $t \leftarrow 0$,
2. **Sélectionner** n individus dans $P(t+1)$ avec remise suivant une distribution de probabilité proportionnelle à l'évaluation des individus par f et leur appliquer les opérateur de *croisement* et de *mutation* avec des probabilités respectives de p_{cross} et p_{mut}.
3. **Évaluer** les individus de $P(t+1)$,
   $t \leftarrow t+1$,
   recommencer le processus de sélection ou arrêter.

Cet algorithme se veut valide pour tout domaine étudié tant qu'il est possible de modéliser les individus de la population par un codage binaire. Cependant, le codage binaire n'est pas toujours bien adapté à la résolution de problèmes par un AG notamment lorsqu'on traite des espaces de recherche assez grands [Whitley et Hanson, 1989]. C'est pour cette raison que les représentations utilisées ont été généralisées aux espaces continus, à des structures arborescentes permettant par exemple de coder des représentations de fonctions pour des problèmes de codage de fonctions mathématiques ou à des individus structurés [Davis, 1991, Koza, 1992b].

On distingue également deux grandes familles de codage : le codage direct pour lequel un individu représente directement une solution au problème posé, et le codage indirect pour lequel un individu exprime une manière de construire une solution au problème. Ce dernier codage peut être utilisé de manière préférentielle car il correspond mieux à la réalité de la nature où les gènes déterminent le phénotype de l'individu qui va permettre à cet individu de dominer dans son environnement. Il permet d'introduire des heuristiques du domaine traité.

Plus précisément, les opérateurs évolutionnaires ont un rôle crucial dans la gestion de la population. Au cours des années qui ont suivi l'apparition des AG, une grande diversité de représentations et d'opérateurs associés ont vu le jour. Nous allons détailler les plus utilisés dans la section suivante.

2.1.2 Opérateurs génétiques

Comme nous l'avons vu précédemment, il existe principalement trois types d'opérateurs agissant sur une population donnée afin de générer sa descendance. L'opérateur de *sélection*, l'opérateur de *mutation* et l'opérateur de *croisement*.

2.1.2.1 Opérateur de sélection

Une fois l'évaluation d'une génération de la population d'individu réalisée, on opère une sélection à partir de la fonction de *fitness*. Cette sélection a pour but de déterminer le potentiel de reproduction des chromosomes de chaque individu. Les éléments passant cette épreuve sont ainsi sélectionnés pour engendrer la génération suivante. Plusieurs techniques de sélection ont vu le jour dans la littérature [Michalewicz, 1996].

La méthode la plus couramment utilisée est certainement celle de la roue de Monte Carlo dans laquelle chaque individu voit sa probabilité de sélection proportionnelle à sa *fitness* - valeur obtenue par la fonction de *fitness*.

Cette solution a l'inconvénient d'accélérer plus qu'il ne le faut la convergence de l'AG et de le limiter à une certaine portion de l'espace des solutions. La pression de sélection est trop forte ou trop uniforme. Il est en effet nécessaire de maintenir une diversité génétique suffisante dans la population afin de constituer un réservoir de gènes pouvant être utiles par la suite. Une sélection trop élitiste peut éliminer certains gènes qui, une fois combinés avec d'autres, peuvent se révéler intéressants.

La sélection par rang est elle aussi très utilisée [Whitley, 1989]. Elle consiste à ordonner les individus dans un ordre décroissant de leur *fitness* et à leur associer une probabilité de sélection dépendant uniquement de leur rang.

Une solution de sélection alternative intéressante est la sélection par tournoi. Elle consiste à choisir aléatoirement k individus dans la population et à les confronter entre eux - grâce à la fonction d'évaluation - afin de ne garder que le meilleur. Cette opération est renouvelée autant de fois que nécessaire afin d'obtenir le nombre d'individus désiré. Il est tout à fait possible que certains individus participent à plusieurs tournois. S'ils

gagnent plusieurs fois, ils seront sélectionnés plusieurs fois, ce qui favorisera la pérennité de leurs gènes. Cette méthode permet une plus grande conservation de la diversité des gènes dans la population - tant que k ne s'approche pas trop de la taille de la population - tout en favorisant les individus de plus forte *fitness*.

Sélection multicritère Cette dernière méthode de sélection permet également de réaliser une optimisation multi-critères. Le problème posé consiste à sélectionner les individus mais en tenant compte de plusieurs critères au lieu d'un seul. On considère que la technique basique consistant à réunir tous les critères sous la forme d'une somme pondérée ne répond pas correctement à ce problème. Il faut donc adapter les opérateurs de sélection. Une première méthode [Todd et Sen, 1997] consiste à sélectionner tour à tour les individus sur chacun des critères. Mais il est encore plus efficace d'utiliser la notion de Pareto-optimalité (voir section 3.3.2). Cette notion de pareto-optimalité permet d'établir une relation de dominance entre les individus portant sur plusieurs critères. Dans un tournoi binaire, où seuls deux individus se confrontent, l'individu finalement sélectionné sera celui qui domine l'autre au sens de pareto. L'ordre induit par l'optimalité selon Pareto étant partiel, il arrive qu'il soit impossible de décider si un individu domine l'autre. Dans ce cas, il faut utiliser des techniques plus avancées [Zitzler, 2001, Deb et Kalyanmoy, 2001]. On peut choisir par exemple de ne sélectionner que les individus non-dominés. Cependant, s'il est nécessaire de fixer un ordre sur les individus, alors plusieurs approches sont possibles. Par exemple, on peut utiliser une dominance de rang pour laquelle on prend en compte le nombre d'individus dominant un individu. D'autres approches utilisent la profondeur de dominance, dans ce cas la population est divisée en plusieurs fronts, chacun correspondant à l'ensemble des individus n'étant dominé par aucun autre. Enfin une autre technique consiste à utiliser le nombre d'individus dominés par un certain individu. Les individus non dominés peuvent alors être ordonnés entre eux. Nous verrons au chapitre 3 que nous introduisons une heuristique spécifique à notre problème pour gérer la sélection multi-critère, heuristique qui aura l'avantage de laisser l'utilisateur fixer l'importance des différents critères. Il s'agira de donner des poids à chacun des critères, et d'utiliser ces poids en cas de non dominance entre deux individus.

2.1.2.2 Opérateur de croisement

Le phénomène de croisement est une propriété naturelle de l'ADN. C'est par analogie qu'ont été conçus les opérateurs de croisement dans les AG.

Cette famille d'opérateurs a pour but de répartir aléatoirement les individus sélectionnés en couples afin d'engendrer une nouvelle génération. Cette génération est obtenue en copiant et recombinant les deux chromosomes parents de manière à générer deux chromosomes enfants possédant des caractéristiques issues des deux parents.

L'opérateur de croisement favorise l'exploitation de l'espace de recherche en examinant de nouvelles solutions à partir de deux solutions actuelles. En effet, cet opérateur assure le brassage du matériel génétique de la population et l'accumulation des gènes

favorables en les multipliant. Ce brassage permet alors de créer de nouvelles combinaisons de gènes qui peuvent se révéler potentiellement favorables.

Le croisement *à un point* est l'opérateur de croisement historique des AG. Il consiste à choisir au hasard un point de croisement pour chaque couple d'individus sélectionné. Ce point divise le génome en deux sous-chaînes. On échange ensuite les deux sous-chaînes terminales de chacun des deux chromosomes, ce qui produit deux enfants. La figure 2.2 illustre ce fonctionnement.

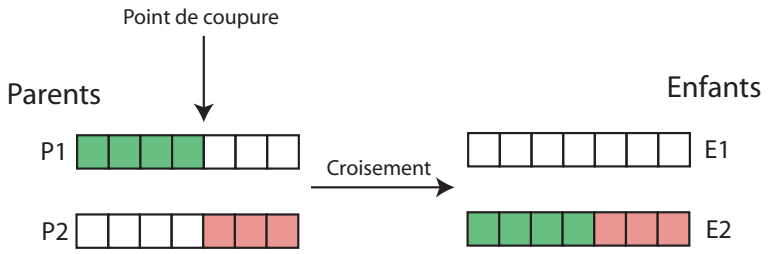


FIG. 2.2 – Opérateur de croisement à 1 point.

Cependant, le croisement à un point ne se révèle réellement efficace que dans un nombre limité de cas. De meilleurs résultats sont obtenus en étendant ce principe en utilisant plusieurs points de coupures de chromosomes [Spears et De Jong, 1991]. Cette opération peut même être généralisée jusqu'au cas limite correspondant à un croisement uniforme [Syswerda, 1989]. Dans ce cas, chaque gène des deux individus parents est échangé avec une probabilité donnée. Ce modèle peut même être généralisé avec des algorithmes comme PBIL qui utilise un vecteur de probabilités pour générer chaque individu [Baluja, 1994].

2.1.2.3 Opérateur de mutation

L'opérateur de mutation consiste à modifier un gène sélectionné dans un chromosome et à le remplacer par une valeur aléatoire. La probabilité de mutation d'un gène est généralement très faible dans un environnement stable afin de favoriser la capitalisation des gènes favorables tout en permettant d'élargir l'espace des solutions évaluées. Cependant, dans un environnement dynamique subissant une évolution rapide, il peut être judicieux d'employer un taux de mutation relativement élevé permettant une adaptation rapide de la population. Le taux de mutation est ainsi à adapter en fonction de l'application et du type de mutation employé.

L'opérateur de mutation apporte aux AG la propriété d'ergodicité de parcours de l'espace [Fogel *et al.*, 1966]. Cette propriété indique que l'AG est capable d'atteindre tous les points de l'espace des solutions, sans pour autant avoir la nécessité d'énumérer l'ensemble de points de l'espace. D'un point de vue théorique, les propriétés de convergence des AG sont fortement dépendantes de cet opérateur. Il garantit ainsi que

l'optimum global peut être atteint.

Les AG se prêtent facilement à la parallélisation du fait de leur conception. Nous allons aborder dans la section suivante les différentes méthodes de parallélisation des AG présentes dans la littérature. À ce sujet, on peut consulter deux états de l'art des techniques existantes de parallélisation dans les méta-heuristiques dont nous relatons l'essentiel dans la suite de ce chapitre [Crainic et Toulouse, 1998, Crainic et Toulouse, 2003].

Nous utiliserons la parallélisation dans les méta-heuristiques de recherche d'information que nous avons étudiées afin d'accélérer grandement le temps nécessaire à leur exécution.

2.1.3 Parallélisation des algorithmes génétiques

Le développement des réseaux et des machines parallèles a amené plusieurs auteurs à s'intéresser à la parallélisation des AG. L'intérêt de cette évolution réside principalement dans le gain en temps de calcul, mais également dans la définition de nouveaux algorithmes. De nombreux modèles ont été proposés dans la littérature depuis le début des recherches sur les AG mais on recense principalement deux types de parallélisation.

La première concerne la parallélisation de l'évaluation des individus. Elle a été introduite pour accélérer les traitements des calculs qui peuvent s'avérer coûteux dans une méta-heuristique, sans essayer d'explorer d'avantage l'espace de recherche. Généralement cette méthode correspond à une méthode séquentielle dont les traitements ont été accélérés.

Dans l'autre cas, la recherche est divisée en plusieurs entités avec différents niveaux de synchronisation et de coopération. Le but de cette méthode est d'explorer plus activement l'espace de recherche en initialisant l'algorithme sur plusieurs points de cet espace. Les recherches s'effectuent ensuite simultanément à partir de ces points initiaux.

2.1.3.1 Parallélisation au niveau de l'évaluation des individus

L'idée principale de cette première méthode est de conserver sur une unique machine la population totale gérée par l'AG parallèle. On considère ici une machine *maître* et plusieurs machines *esclaves* dans un ensemble synchronisé. La machine *maître* est chargée d'effectuer toutes les opérations génétiques sur la population : sélection, croisement et mutation. Les opérations de calcul potentiellement plus importantes comme l'évaluation des individus sont alors confiées à des machines ou à des processus - dans le cas de l'utilisation de machines à plusieurs processeurs - *esclaves*. Dès réception de la totalité des évaluations par les machines *esclaves*, la machine *maître* peut alors procéder à la constitution de la génération suivante.

Cette stratégie a été introduite par Grefenstette et est probablement la première tentative de parallélisation d'une méta-heuristique [Grefenstette, 1981]. Une seule population est donc considérée et il n'y a pas de restriction particulière à appliquer sur les opérateurs de mutation et de sélection.

Il faut noter que les mécanismes mis en jeu demandent un grand nombre de communications pour envoyer les données et les évaluations dans un système à mémoire distribuée. Ce problème s'amplifie avec le nombre d'entités parallélisées. Fogarty et Huang [Fogarty et Huang, 1991] ont implémenté un tel système et ont conclu que le temps de communication devient négligeable du moment que le temps mis pour évaluer un individu est important.

En implémentant ce type d'algorithme parallèle sur une machine à mémoire partagée pour résoudre un problème d'emploi du temps, Ambremson et Abela ont observé une progression linéaire des performances avec une augmentation des ressources mises à disposition [Abramson et Abela, 1991]. Cependant dans cette approche, le nombre de ressources de calcul disponibles est limité par l'architecture matérielle utilisée.

Pour concilier les avantages des deux méthodes précédentes, Hauser et Männer [Hauser et Männer, 1994] utilisent plusieurs systèmes à mémoire partagée communiquant ensemble. Les performances observées sont alors relativement bonnes et offrent un compromis raisonnable entre rapidité de traitement et nombre de ressources mises en jeu.

Ce type de stratégie de parallélisation reste limité à la nature du problème ; la méthode n'est donc intéressante que si le temps passé pour effectuer l'évaluation d'un individu est important. Le second type de parallélisation existant communément dans les AG considère plusieurs populations d'individus gérées de manière plus ou moins asynchrone.

2.1.3.2 Parallélisme par sous-populations d'individus

Contrairement à la méthode précédente, cette deuxième approche fait fonctionner plusieurs algorithmes génétiques en parallèle sur des populations d'individus réduites. Chaque machine exécutant un AG fait évoluer sa population indépendamment des autres. Selon un critère défini, par exemple une probabilité fixée à l'avance, une machine peut décider de rassembler ses meilleurs individus pour en transmettre une certaine quantité à une autre machine de son choix. La machine réceptrice intègre alors ces nouveaux éléments dans sa propre population en éliminant les moins bons de ses individus.

En procédant de cette manière, chaque AG a la possibilité de converger vers des optima qui seront différents de ceux calculés sur les autres machines. De plus, cela offre la possibilité de traiter des populations d'individus de très grandes capacités tout en donnant des résultats dans un temps raisonnable.

Ce système de parallélisation a été très largement étudié dans les AG. On retrouve une quantité considérable de variations sur ce thème que l'on peut séparer en deux catégories : à gros grain et à grain fin.

Parallélisme à gros grains (*coarse grain*) Dans cette approche, on utilise généralement le même AG pour chaque sous-population d'individus mais des recherches ont

été menées (notamment dans [Schlierkamp-Voosen et Mühlenbein, 1994]) afin d'étudier différentes stratégies d'exploration.

Le parallélisme à gros grain ajoute des opérateurs et des règles gérant la communication entre les différentes sous-populations. Un opérateur de migration est ajouté afin de permettre l'échange d'informations entre les sous-populations. Une politique d'émigration détermine quelle stratégie (hasard, meilleur individu, etc) employer pour sélectionner les individus candidats à la migration. Une politique d'immigration gère le remplacement d'individus dans la population d'accueil à l'arrivée d'un nouvel élément. D'autres paramètres gèrent la fréquence d'échange ou encore le nombre de départs simultanés d'une population.

Les études menées par Tanese [Tanese, 1987] et Belding [Belding, 1995] ont montré d'une part que sans la connaissance des meilleures combinaisons de paramètres possibles, des méthodes parallèles utilisant différentes valeurs de paramètres permettent d'obtenir de meilleurs résultats. D'autre part, ils ont montré qu'un fort taux de migration diminuait sensiblement le temps nécessaire à la résolution d'un problème. Le modèle *island* (parallélisme par îlots), initié par Cohoon a permis d'étudier le rôle de la migration au niveau de l'évolution de la qualité de la population [Cohoon *et al.*, 1987]. Dans ce modèle les individus peuvent migrer dans n'importe quelle autre population. Il a été observé que les nouvelles solutions apparaissent majoritairement rapidement après l'introduction de nouveaux individus dans une population. Les AG parallèles par îlots ne sont pas nécessairement équivalents aux AG séquentiels décrits précédemment car l'application des opérateurs de croisement et de mutation est localisée dans un voisinage donné de la population totale.

Mühlenbein introduit en 1991 un nouveau modèle, le *stepping-stone*, dans lequel les échanges ne peuvent se faire qu'entre populations d'individus voisines dans l'espace de recherche [Mühlenbein, 1991]. Son originalité vient de l'utilisation d'une heuristique de type *hill-climbing* lorsque la qualité d'une population ne s'améliore pas au bout d'un certain nombre d'itérations. Cette méthode a montré sa grande efficacité et a beaucoup contribué au développement des AG parallèles.

Dans un autre registre n'utilisant pas de migration entre les sous-populations, Whitley a constaté qu'en adaptant le paramètre de mutation en fonction de ceux obtenus par la sous-population obtenant les meilleurs résultats, les performances s'améliorent sensiblement [Whitley et Starkweather, 1990].

Parallélisme à grains fins (*fine grain*) Ce sont des méthodes asynchrones qui divisent la population en un nombre conséquent de sous-éléments. Ces sous-éléments peuvent aller du cas limite (décrit dans [Spiessens et Manderick, 1991]) pour lequel un seul individu est représenté par un processeur. Chaque entité est connectée à plusieurs autres entités dans son voisinage formant un groupe et les opérateurs génétiques sont appliqués à ce groupe grâce à des échanges entre les individus le composant.

Ces méthodes sont souvent identifiées comme des méthodes massivement parallèles car chaque individu est représenté idéalement par un seul processeur. Whitley qualifie même ce type d'algorithme dans [Whitley, 1993] de modèle cellulaire car on retrouve un parallèle avec les automates cellulaires avec des règles de réécritures probabilistes et

un alphabet composé des éléments de l'espace de recherche.

Ces techniques ont été implémentées notamment par Spiessens et Manderick dans [Spiessens et Manderick, 1991] où chaque individu est mis à jour sur une architecture massivement parallèle.

Peu d'auteurs ont comparé les performances respectives d'une parallélisation à grains fins et à gros grains et les résultats sont souvent contradictoires comme le souligne Cantù-paz dans [Cantú-Paz, 1999] et [Cantú-Paz, 2000].

Cette méthode de parallélisation demande cependant de veiller à respecter certaines contraintes afin de garantir une certaine fiabilité de déroulement des AG. La plus importante d'entre elles étant sûrement de ne pas subdiviser la population en de trop petites sous-populations.

La parallélisation est extrêmement efficace pour accélérer les temps de résolution des AG. Mais il convient d'étudier le type de problème rencontré afin de bien sélectionner la méthode à utiliser. Le ratio entre le temps mis pour l'évaluation d'un individu et l'exécution d'un opérateur génétique est un bon indicateur permettant d'effectuer un choix judicieux. Dans le cas d'un temps d'évaluation important, la parallélisation des calculs est préférable à une parallélisation par îlots, et réciproquement.

Dans la suite, nous décrivons deux autres approches : la recherche tabou et les algorithmes à base de populations de fourmis. Nous utiliserons et comparerons dans les chapitres suivants des adaptations de ces méthodes ainsi que des AG à la résolution du problème de recherche d'information.

2.2 La recherche Tabou

La recherche tabou est une méthode d'optimisation mathématique de la famille des techniques de recherche locale présentée par Fred Glover en 1986 [Glover, 1986]. L'idée de base consiste à introduire la notion d'historique dans la politique d'exploration des solutions afin de diriger au mieux la recherche dans l'espace. Cette méthode s'est révélée particulièrement efficace et a été appliquée avec succès à de nombreux problèmes difficiles [Glover et Laguna, 1998].

2.2.1 Méthode originale

Cette méthode a été baptisée *recherche tabou* par Fred Glover pour exprimer l'interdiction de reprendre des solutions récemment visitées. À cette fin, des structures de mémoire adaptées sont mises en œuvre, ce qui permet à la recherche tabou d'augmenter les performances d'une méthode de recherche locale.

Classiquement, une méthode de recherche telle que l'ascension locale passe à côté de certaines zones de l'espace de recherche qui pourraient contenir un optimum global. La recherche tabou va tenter de modifier petit à petit la structure du voisinage d'un point de l'espace de recherche afin de contraindre l'algorithme à explorer d'autres régions de

l'espace. D'un point de vue formel, le voisinage d'une solution s noté $V(s)$ est modifié en $V^*(s)$ en fonction de la mémoire contenant les derniers pas parcourus dans l'espace des solutions. La recherche tabou utilise une procédure locale à partir de $V^*(s)$ pour aboutir à une nouvelle solution s'. Le processus est ensuite réitéré à partir de s' jusqu'à ce qu'un critère d'arrêt soit satisfait.

De cette manière, un mouvement permet de passer d'une solution valide à une autre. On dit alors que la nouvelle solution est un *voisin* de la précédente et on appelle *voisinage* l'ensemble des solutions que l'on peut atteindre à partir d'une solution.

De manière plus détaillée, chaque point exploré est enregistré dans une mémoire de taille fixe. À chaque itération, tous les mouvements possibles sont examinés et le meilleur, ou plus exactement le moins mauvais est sélectionné parmi tous les points du voisinage direct de la position courante ne figurant pas dans la mémoire. Cette modification du paysage local de l'espace de recherche permet de plus d'éviter des mouvements cycliques, répétant indéfiniment la même suite de déplacements. La mémoire des points visités est appelée *liste tabou* du fait que les derniers mouvements effectués sont considérés comme interdits (ou tabous). Le mouvement effectivement choisi à chaque itération est donc le meilleur mouvement non tabou.

Lorsque la mémoire est pleine, elle est gérée comme une liste circulaire en FIFO - *First In First Out* - : on élimine le plus vieux point tabou et on insère la nouvelle solution. La taille de la mémoire permet de ne pas saturer rapidement les ressources disponibles pour la recherche et permet de surcroît d'adapter facilement la méthode à un espace de recherche dynamique (comme l'est par exemple Internet).

Cette méthode a par ailleurs l'avantage d'être facilement paramétrable. Il existe en effet au maximum deux paramètres qui peuvent réellement influencer la recherche. Le premier consiste à bien choisir la taille de la liste tabou. Elle est généralement déterminée empiriquement et varie avec les problèmes, mais c'est une donnée primordiale : une taille trop petite peut amener l'algorithme à boucler sur une zone locale de l'espace de recherche tandis qu'une trop grande taille saturera rapidement les ressources disponibles. Le deuxième se retrouve relativement fréquemment dans les méthodes d'optimisation utilisant une exploration locale afin de mener à l'optimum : le choix du critère d'arrêt. Ce dernier peut s'avérer difficile à déterminer pour éviter de ne prolonger trop longtemps la recherche ou de rater l'optimum global recherché.

À partir de cet algorithme initial, certaines adaptations ont été élaborées. Ces améliorations ont été introduites afin de pallier à des problèmes constatés dans l'analyse de l'exploration de l'espace de recherche. [Glover et Laguna, 1998] recense toutes ces techniques.

2.2.2 Diverses améliorations

L'utilisation de points tabous peut empêcher, dans certains cas, la méthode tabou d'atteindre une solution intéressante. Par conséquent, il est nécessaire de ne pas être trop restrictif tout en évitant de boucler dans le parcours d'une branche dans

la recherche d'une solution. Le *critère d'aspiration* a été introduit par Glover dans [Glover et Laguna, 1993] à cet effet et consiste à enlever le statut tabou associé à une transformation si celle-ci permet d'aboutir à une solution meilleure que toutes les solutions trouvées jusqu'alors. Mais ce critère ne se limite pas à ce cas particulier et il est également possible d'utiliser une fonction d'aspiration dont le but est de toujours aller d'une solution à une solution meilleure.

Dans certains cas, il peut être également judicieux d'intensifier les recherches sur des zones qui paraissent prometteuses. Par exemple, [Chakrapani et Skorin-Kapov, 1993] utilise cette technique en repartant de la meilleure solution avec une liste tabou vide. Cet examen approfondi peut permettre de dégager quelques propriétés communes définissant les régions intéressantes de l'espace de recherche. Il est alors aisé d'orienter la recherche vers ces zones en rendant tabou tous les points menant à sortir de ces régions.

D'une manière symétrique, lorsque le processus de recherche parcourt une branche sur une longue période, il est possible de le stopper et de diversifier la recherche sur une autre zone de l'espace. L'algorithme reprend alors généralement sur une autre solution générée aléatoirement. Mais il est possible d'utiliser une stratégie plus fine en mémorisant les solutions les plus fréquemment visitées et en imposant un système de pénalités, afin de favoriser les mouvements les moins souvent utilisés.

La taille de la liste tabou prise en compte peut également évoluer au cours du temps de manière dynamique [Battiti et Tecchiolli, 1994]. Cette méthode adapte la taille de la liste tabou lorsque des cycles ont été détectés dans le parcours de l'espace de recherche par l'algorithme tabou.

Enfin, on retrouve dans la littérature principalement deux politiques de détermination du meilleur voisin : la politique du *Best-Fit* qui consiste à sélectionner le meilleur voisin non tabou, et la politique du *First-Fit* qui consiste à sélectionner le premier voisin qui satisfait les contraintes tabous. L'utilisation de cette dernière politique est souvent réservée à des problèmes dans lesquels la taille du voisinage ne permet pas d'effectuer une évaluation complète.

2.2.3 Algorithme

L'algorithme 2.1 définit le fonctionnement canonique de la méthode tabou. Dans le formalisme utilisé, S_{opt} représente la solution optimale courante détectée, et $Voisin(S)$ représente une fonction d'exploration locale examinant le voisinage direct du point S.

2.2.4 Parallélisation

Nous avions rencontré deux types de parallélisation dans les AG (voir section 2.1.3). On retrouve ces mêmes concepts dans la recherche tabou avec une troisième notion : la décomposition d'un problème en plusieurs sous-problèmes. Dans ce dernier type de parallélisation, chaque sous-problème est résolu indépendamment et la solution globale peut alors être construite.

```
(1)  Initialiser S de manière aléatoire
(2)  S_opt ← S
(3)  Ajouter S à la liste tabou
(4)  tantque La condition d'arrêt n'est pas vérifiée faire
(5)     répéter
(6)        S' = Voisin(S)
(7)     tantque S' ∈ liste tabou fin
(8)     si la liste tabou est pleine alors
(9)        Remplacer le dernier élément de la liste tabou par la solution S'
(10)    sinon
(11)       Ajouter S' à la liste tabou
(12)    finsi
(13)    si S' est meilleur que S_opt alors
(14)       S_opt ← S'
(15)    finsi
(16)    S ← S'
(17) fintantque
```

ALG 2.1: Algorithme de recherche Tabou

2.2.4.1 Parallélisme bas-niveau

Dans ce type de parallélisation, on considère une machine *maître* qui exécute un algorithme tabou séquentiel. Les mouvements possibles dans le voisinage du point courant sont évalués à chaque itération en parallèle sur des machines *esclaves*. Le processus *maître* reçoit ensuite toutes les évaluations effectuées par les processus *esclaves* qui lui permettent de prendre une décision quant au chemin à suivre pour la prochaine itération de l'algorithme. Cette technique a notamment été employée dans [Crainic *et al.*, 1997], [Taillard, 1993] et [Chakrapani et Skorin-Kapov, 1993].

Dans ce dernier article, Chakrapani et Skorin-Kapov rapportent que cette méthode de parallélisation est très efficace et que l'augmentation du nombre de ressources mises à disposition diminue de manière quasi linéaire le temps de calcul nécessaire à l'obtention de l'optimum.

2.2.4.2 Parallélisation par décomposition de l'espace de recherche

Ce type de parallélisation est très utilisé dans la recherche tabou. Elle prend en considération plusieurs algorithmes tabou parcourant l'espace de recherche à partir de points initiaux différents ou en utilisant des stratégies différentes. On peut décomposer l'ensemble de ces algorithmes en trois catégories : les algorithmes 1) utilisant un même point de départ mais des stratégies de recherche différentes [Battitti et Tecchiolli, 1992], 2) utilisant plusieurs points de départ avec la même stratégie d'exploration de l'espace de recherche [Rego et Roucairol, 1992] et 3) utilisant à la fois des points initiaux et des

stratégies différents [Crainic *et al.*, 1997].

Les algorithmes tabou en concurrence peuvent également agir de manière totalement indépendante - la solution globale étant déterminée à la fin - ou au contraire s'échanger des informations grâce à des communications synchrones. Les méthodes asynchrones ont toutefois été étudiées dans la littérature beaucoup plus profondément.

2.2.4.3 Parallélisation en sous-problèmes

Pour effectuer ce type de parallélisation en recherche tabou, on décompose typiquement le problème en autant de sous-problèmes que de variables de décision. Pour chaque sous-problème une recherche tabou indépendante des autres va chercher à optimiser la variable concernée.

Le problème évident lié à cette méthode réside dans sa dépendance à la nature des problèmes testés. Elle a pu aussi bien obtenir de très bons résultats dans certaines applications dans [Rochat et Taillard, 1995] ou [Porto et Ribeiro, 1995] que se montrer relativement peu performante dans [Crainic *et al.*, 1995].

Enfin, une autre voie de parallélisation (décrite dans [Preux et Talbi, 1999]) consiste à employer plusieurs méta-heuristiques de natures différentes coopérant à la résolution d'un même problème. Cette technique se rapproche de l'intelligence artificielle distribuée que nous aborderons dans la section suivante. L'intérêt majeur n'est plus ici d'accélérer les traitements mais d'avoir des activités concurrentes. On peut voir par exemple à cet effet [Bachelet *et al.*, 1998] dans lequel un algorithme génétique cherche des solutions de départ à un algorithme tabou dans des zones de l'espace de recherche encore peu explorées.

Dans la section suivante nous abordons un autre type de méta-heuristiques que nous utiliserons également dans le chapitre 6 pour la construction d'une méthode de recherche d'information sur Internet. Il s'agit des approches à base d'agent et plus particulièrement des approches s'inspirant du comportement observé dans les colonies de fourmis.

2.3 Introduction aux algorithmes multi-agents et à base de fourmis

2.3.1 Les algorithmes multi-agents

La notion de système multi-agents est apparue dès lors que l'intelligence artificielle a montré ses limites en ce qui concerne la capacité d'un unique *super-individu* à résoudre des problèmes complexes. L'émergence d'une nouvelle discipline vers la fin des années 1970, l'Intelligence Artificielle Distribuée (IAD), a permis de voir les problèmes sous une autre forme [Hewitt *et al.*, 1973, Hewitt, 1977]. À cette époque, Hewitt chercha une nouvelle manière de résoudre des théorèmes en considérant plusieurs entités actives appelées acteurs. La résolution consiste alors à confronter plusieurs points de vue, chacun émanant d'une entité acteur. Le plus souvent, il est nécessaire de décomposer

un problème complexe en tâches plus simples à résoudre. Ces tâches peuvent ainsi être traitées par des systèmes moins lourds à concevoir et plus faciles à maîtriser.

L'IAD est née de l'apparition de systèmes multi-experts dont les objectifs sont similaires à l'idée d'éclatement d'une tâche à accomplir en plusieurs sous-tâches plus simples à résoudre par plusieurs entités spécialisées. Cependant, la gestion de plusieurs entités coopérant afin de résoudre un problème commun a généré d'autres interrogations et problématiques. Par exemple, l'indépendance des différentes entités pose des questions au niveau de la mise en œuvre de systèmes de communication comprenant une coopération des *agents*, des systèmes de négociation et de synchronisation. La gestion de toutes ces interactions a conduit à la notion de systèmes multi-agents.

Ces problèmes se rencontrent relativement fréquemment dans la nature. C'est pourquoi ce domaine s'intéresse à la sociologie ou à l'éthologie des espèces sociales, très concernées par les phénomènes de groupe. L'étude des choix faits par les mécanismes d'évolution, par exemple pour les espèces sociales comme les fourmis, les abeilles ou les étourneaux, permet de dégager trois grands axes dans l'étude de l'organisation des systèmes multi-agents co-évoluant : la coordination spontanée d'agents simples permettant à un système multi-agents d'avoir un comportement global complexe du point de vue d'un observateur extérieur, la communication d'un savoir au sein d'une communauté d'agents et l'échange d'informations entre agents hétérogènes d'une même communauté.

Les systèmes multi-agents sont un domaine relativement jeune. On peut considérer qu'ils sont apparus dans les années 80 grâce notamment aux travaux de Erman et Lesser sur le système Hearsay [Erman *et al.*, 1980]. Ces auteurs ont proposé l'idée du *tableau noir* jouant le rôle de transmetteur d'informations entre les différents agents. Ce système permet à tout agent d'un système de venir consulter le tableau noir et d'échanger ainsi des informations avec ses congénères.

Les années 90 ont vu se développer les systèmes multi-agents et un certain formalisme est apparu. Dans [Ferber, 1995], Ferber a ainsi décrit un agent comme une entité physique ou virtuelle possédant plusieurs propriétés. Cet organisme doit pouvoir :

- être capable d'agir dans un environnement,
- pouvoir communiquer directement avec d'autres agents,
- être mu par un ensemble de tendances, par exemple par une fonction de satisfaction qu'il cherche à optimiser,
- posséder des ressources propres,
- être capable de percevoir une partie de son environnement,
- posséder des compétences et offrir des services à d'autres agents,
- pouvoir éventuellement se reproduire.

Plusieurs familles d'algorithmes multi-agents ont vu le jour. La plus grande source d'inspiration reste cependant l'observation de la nature. Chacune de ces méthodes essaye ainsi de tirer parti d'un comportement observé dans les groupes sociaux. Les algorithmes biomimétiques s'inspirant du comportement social observé chez les fourmis en sont un bon exemple.

2.3.2 Particularité des approches à base de fourmis

En observant les insectes sociaux on s'aperçoit qu'une communauté d'individus simples peut présenter au niveau collectif des capacités supérieures à la somme des capacités individuelles. Les fourmis en sont notamment une parfaite illustration : individuellement, une fourmi n'est pas considérée comme une créature très intelligente mais prise dans sa globalité, une colonie de fourmis est capable d'accomplir des tâches complexes et faire preuve d'une intelligence étonnante [Hölldobler et Wilson, 1990]. Elles sont capables d'identifier rapidement le chemin le plus court menant à la nourriture ou encore de s'entraider spontanément afin de pouvoir traîner un très gros morceau de nourriture qu'une seule fourmi est incapable de transporter. Les fourmilières sont d'ailleurs très bien organisées et sont des constructions assez élaborées qui peuvent comporter plusieurs millions d'individus.

Généralement, chaque fourmi est spécialisée dans une tâche particulière. On retrouve ainsi dans une fourmilière plusieurs *familles* de fourmis, certaines s'occupant des larves, d'autres ramenant de la nourriture ou des matériaux de construction de la fourmilière ou encore d'autres défendant le reste de la colonie. Mais cette spécialisation n'est pas figée et s'il manque des individus pour accomplir une tâche particulière, des fourmis appartenant habituellement à une *famille* différente peuvent venir aider leurs congénères et accomplir des tâches qui ne leur sont pas généralement attribuées.

La plupart des interactions entre les individus d'une colonie de fourmis sont réglées par des substances chimiques : les phéromones. Ces éléments chimiques interviennent à plusieurs stades dans la vie de la communauté des fourmis. Grâce à elles, les fourmis peuvent retrouver leur chemin, ou encore émettre une alarme. Les phéromones permettent aussi de changer la spécialité d'une fourmi. Chacune possède un seuil de phéromone pour chaque spécialité et n'effectue une tâche donnée qu'à partir du moment où le taux de phéromones correspondant à cette tâche dépasse ce seuil. Ainsi, lorsque les larves ont besoin de fourmis nourricières pour les nourrir, elles émettent des phéromones qui vont attirer d'autres fourmis se spécialisant à leur tour dans la nourriture des larves.

Les algorithmes à base de fourmis artificielles s'inspirent du comportement collectif des fourmis pour résoudre des problèmes d'optimisation combinatoire ou de classification. On peut considérer que les fourmis sont des agents qui agissent sur un environnement donné. Par exemple, dans une colonie de fourmis, une ouvrière est autonome et aide la communauté en rapportant de la nourriture. Elle communique avec d'autres ouvrières ne trouvant pas de nourriture en laissant une trace de phéromone menant à la source de nourriture afin de transmettre son savoir. Enfin, elle est également capable d'émettre un signal d'alarme lorsqu'elle rencontre une fourmi d'une autre colonie pour alerter d'autres fourmis spécialisées dans le combat afin de protéger la colonie.

Dans les sections suivantes, nous examinerons les algorithmes ayant prouvé le potentiel et l'efficacité d'algorithmes s'inspirant du comportement des individus d'une colonie de fourmis. Ces exemples nous serviront de base pour l'élaboration d'une approche biomimétique pour la résolution du problème de recherche d'information sur Internet.

2.3.3 Optimisation par algorithme à base de fourmis

Nous allons détailler dans cette section deux algorithmes particuliers inspirés de l'observation des fourmis réelles. Le premier (ACO) est à l'origine du développement des approches à base de fourmis et a montré son efficacité sur des problèmes classiques rencontrés en optimisation. Il prend en compte les interactions physico-chimiques entre les individus d'une colonie de fourmis permettant de s'adapter à leur environnement de manière dynamique. Dans un deuxième temps, nous examinerons l'algorithme API qui s'inspire du comportement de fourragement d'une espèce de fourmis plus primitive afin de rechercher l'optimum d'une fonction donnée et qui a servi de base à la construction de l'algorithme *Antsearch* décrit dans le chapitre 6.

2.3.3.1 L'algorithme fondateur : ACO

La première démonstration de la capacité et de la performance des algorithmes à base de fourmis artificielles a été réalisée par Dorigo en 1992 au travers de l'heuristique *Ant Colony Optimization* (ACO) [Dorigo, 1992]. Cette métaheuristique s'inspire du comportement de fourragement observé chez les fourmis réelles en tenant compte des interactions physiologiques entre les fourmis.

Ces interactions permettent aux fourmis de résoudre naturellement des problèmes complexes comme par exemple la découverte du plus court chemin entre les différentes sources de nourriture et le nid de la colonie. L'algorithme tient compte de la communication chimique observée chez les fourmis réelles. Lorsqu'une fourmi de la colonie marche d'une source de nourriture au nid, elle dépose une substance chimique sur le sol en proportion avec le potentiel de la source de nourriture. Cette substance est appelée phéromone et attire les autres fourmis de la colonie. Ainsi, lorsque plusieurs possibilités de chemin s'offrent à un élément de la colonie, ce dernier choisira avec une plus grande probabilité les directions marquées par de plus fortes concentrations de phéromone. En procédant de cette manière, les sources de nourriture les plus exploitables seront favorisées par la colonie afin de maximiser ses chances de survie. Ce comportement est à la base de la coopération et la communication menant à la résolution de problèmes classiques comme le problème du plus court chemin ou le problème du voyageur de commerce.

Le premier système ACO introduit par Dorigo est appelé *Ant System* (AS). La description de son fonctionnement nous permet d'établir les fondements des algorithmes de fourmis au travers d'un exemple de résolution de problème classique en optimisation. Cet algorithme a initialement été appliqué à la résolution du problème du voyageur de commerce (PVC) en se basant sur le système de dépose de phéromone observé chez les fourmis.

Le PVC, ou *Traveling Salesman Problem* se définit de la manière suivante :

Définition 1 *Un voyageur de commerce doit visiter un ensemble* $\{v_1, \ldots, v_n\}$ *de* n *villes dont on connaît les distances respectives* $d(v_i, v_j), \forall (i,j) \in \{1, \ldots, n\}^2$. *Le problème consiste à trouver la permutation* σ *de* $\{v_1, \ldots, v_n\}$ *telle que la séquence* $s =$

$\left(v_{\sigma(1)}, \ldots, v_{\sigma(n)}\right)$ *minimise la distance totale* $D(\sigma)$ *parcourue par le voyageur :*

$$D(\sigma) = \sum_{i=1}^{n-1} d\left(v_{\sigma(i)}, v_{\sigma(i+1)}\right) + d\left(v_{\sigma(n)}, v_{\sigma(1)}\right) \tag{2.1}$$

□

Il a été démontré que ce problème est NP-difficile en 1979 [Garey et Johnson, 1979]. En effet, l'espace de recherche correspond à l'ensemble des combinaisons possibles des n villes, soit $n!$ combinaisons. Ce problème peut également être considéré comme la recherche d'un circuit hamiltonien de longueur minimale dans un graphe complet pouvant être anti-symétrique dans le cas général ($\exists(i,j)$ tel que $d(v_i, v_j) \neq d(v_j, v_i)$)

Les fourmis rencontrent ce problème de manière naturelle dans leur environnement lorsqu'elles se déplacent entre les sources de nourriture et leur nid. Elles cherchent en effet à minimiser leur effort qui est très fortement corrélé à la distance qu'elles doivent parcourir. Dans [Bonabeau *et al.*, 1999], les auteurs montrent que l'organisation même de la colonie et les interactions opérant entre les fourmis amènent à la résolution de ce type de problème.

En déposant des phéromones sur les chemins qu'elles empruntent, les fourmis établissent un moyen efficace de résolution du problème du plus court chemin. En effet, lorsque deux chemins de longueurs différentes s'offrent à une colonie de fourmis pour aller du nid à une source de nourriture, les fourmis choisissent initialement avec une probabilité uniforme le chemin à emprunter. Par la suite, les fourmis déposant toutes des phéromones, le plus court des deux chemins va avoir tendance à être davantage marqué incitant les fourmis suivantes à suivre ce même chemin. Ce comportement autoentretenu entraîne la majorité des fourmis à suivre le plus court chemin.

De plus, des expérimentations effectuées sur les fourmis réelles [Goss *et al.*, 1989] ont montré que dès qu'une modification de l'environnement apparaît, entraînant des changements dans la longueur des chemins reliant nourriture et nid, les mécanismes probabilistes mis en jeu dans le comportement induit du déplacement des fourmis permettent à la colonie de s'adapter afin de suivre majoritairement le *nouveau* plus court chemin. Dans la nature, les phéromones s'évaporent petit à petit naturellement, ce qui amplifie davantage encore la capacité d'adaptation des fourmis à une nouvelle situation. Colorni et Dorigo ont montré par ailleurs dans [Colorni *et al.*, 1991] qu'un algorithme s'inspirant de ce comportement est très efficace.

Pour résoudre le PVC, l'algorithme AS étend ces principes en permettant aux fourmis artificielles de sauvegarder quelques données en mémoire et de percevoir une partie de leur environnement. Ces nouvelles facultés vont permettre d'adapter le comportement des fourmis artificielles au problème : chaque fourmi doit se souvenir des villes dans lesquelles elle est déjà passée afin de ne pas y retourner. Dans cette optique, une liste des villes à ne pas visiter est maintenue au sein de la mémoire de la fourmi et

(1) **tantque** Condition de terminaison non rencontrée **faire**
(2) Construction de solutions par les fourmis
(3) Mise à jour des phéromones
(4) Exécuter des actions sur l'environnement
(5) **fintantque**

ALG 2.2: Cadre d'exécution de l'algorithme ACO

réinitialisée une fois qu'un parcours complet a été effectué (i.e. toutes les villes ont été visitées).

La modélisation utilisée considère le graphe orienté $G(A, V)$ représentant l'ensemble des villes considérées dans le problème (les sommets V) et des chemins les reliant (les arcs A). Les fourmis sont disposées sur les sommets du graphe et peuvent voyager de sommet en sommet en suivant les arcs. À chaque voyage d'une ville i à une ville j, elles déposent une trace de phéromone sur l'arête (i, j). Le choix d'une nouvelle ville à parcourir se fait alors en fonction des villes précédemment visitées et selon une probabilité dépendant de la quantité de phéromone présente sur l'arête ainsi que de la distance entre cette ville et la position courante de la fourmi. Une fois que toutes les fourmis ont effectué un parcours complet, le taux de phéromone présent sur chaque arête est mis à jour. Pour cela, le taux présent à l'itération précédente est diminué proportionnellement à une constante d'évaporation définie à l'origine puis renforcée en fonction du parcours des fourmis sur l'arête prise en compte.

L'algorithme se termine au bout d'un temps défini et la solution retenue correspond au chemin de moindre coût - le plus court - parcouru par les fourmis au cours de l'exécution de l'algorithme.

Plusieurs extensions à l'algorithme AS ont vu le jour afin d'améliorer la répartition des phéromones sur le graphe ainsi que d'ajuster plus finement la probabilité de choix d'une ville par les fourmis. Nous ne détaillerons pas ces mises à jour dans ce document mais nous pouvons cependant citer les travaux suivants : [Gambardella et Dorigo, 1995] introduit un système d'apprentissage par renforcement, [Stützle et Hoos, 1997] propose le système $\mathcal{MAX} - \mathcal{MIN}$ AS introduisant des seuils minimum et maximum à la trace de phéromone et [Dorigo et Gambardella, 1997] décrit l'algorithme *Ant Colony System* (ACS). Ce dernier introduit principalement plus de choix probabiliste dans la décision d'exploration d'une ville et correspond à l'heuristique à base de fourmis artificielles apportant les meilleurs résultats pour le PVC.

ACO rassemble l'ensemble de ces algorithmes et peut, d'une manière plus générale, être décrit par l'algorithme 2.2. Les trois opérations peuvent s'exécuter de manière non synchronisée et leur ordonnancement est laissé à la discrétion du programmeur.

La première opération listée consiste à construire une solution au problème grâce à une fourmi en la déplaçant sur les nœuds du graphe représentatif du problème. Ce mouvement est soumis à des décisions locales probabilistes qui tiennent compte à la

fois de la connectivité du graphe et de l'intensité des traces de phéromones déposées par les fourmis dans les précédentes itérations. Au fur et à mesure de son déplacement, la fourmi mémorise la solution qu'elle est en train de construire en terme de chemin parcouru sur le graphe.

La deuxième opération permet d'annoter les arcs du graphe représentatif du problème pour aider les autres fourmis à trouver la meilleure solution. La mise à jour des phéromones peut se faire suivant deux méthodes distinctes : à chaque pas effectué par une fourmi d'une part - on parle alors de mise à jour pas à pas - et une fois la solution construite en suivant le chemin inverse sauvegardé dans la mémoire de la fourmi d'autre part - on parle alors de mise à jour retardée. Il est également important de procéder à l'évaporation de l'ensemble des phéromones déposées sur les arcs du graphe. En effet, l'intensité des traces de phéromones doit décroître au cours du temps afin d'éviter une convergence trop rapide de l'algorithme dans un extremum local. Cette évaporation permet à la colonie de fourmis d'oublier peu à peu ce qu'elle a appris ce qui favorise de plus une meilleure adaptation à tout changement dans l'environnement.

Enfin, il peut être nécessaire d'exécuter certaines actions centralisées qui ne peuvent être effectuées par les fourmis elles-mêmes de manière individuelle. Il peut être par exemple nécessaire de collecter l'ensemble des solutions établies par les fourmis afin de décider s'il est judicieux d'ajouter des phéromones supplémentaires pour aider le processus de recherche en soulignant davantage le chemin menant à la construction de la meilleure solution actuelle. Cette mise à jour de phéromone est appelée mise à jour en différé et permet dans certains cas d'augmenter les performances globales d'un algorithme.

L'algorithme ACO est à la base d'un grand nombre d'applications s'inspirant du comportement collectif des fourmis afin de résoudre un problème donné. Celui-ci s'appuie sur un modèle du comportement de certaines espèces de fourmis. Cependant, toutes les colonies de fourmis n'utilisent pas les mêmes armes et l'évolution des espèces a permis de tester plusieurs stratégies permettant la survie de chaque espèce. Il existe ainsi tout naturellement d'autres modèles algorithmiques utilisés en optimisation et s'inspirant d'espèce de fourmis différentes aux caractéristiques distinctes de celles reprises dans ACO.

Nous présentons dans la section suivante l'algorithme API permettant de ne pas avoir à gérer un système à base de phéromones qui peut se révéler très lourd à maintenir lorsque l'on traite des graphes de très grande dimension comme c'est le cas dans une représentation d'Internet. La modélisation de cet algorithme a servi de base à l'élaboration d'un algorithme de recherche d'information sur Internet présenté dans le chapitre 6.

2.3.3.2 L'algorithme API

L'algorithme API est un algorithme d'optimisation développé à la fin des années 90 s'inspirant de la stratégie de fourragement observé chez les fourmis de la famille *Pachycondyla Apicalis* [Monmarché *et al.*, 2000]. Ces fourmis vivent en Amérique du Sud et plus précisément au Mexique. Elles peuvent être considérées comme assez primitives,

n'utilisent pas de communication poussée à base de phéromone lors de la recherche de nourriture et leur comportement de chasse reste relativement simple. En effet, les fourmis chassent individuellement en tentant de couvrir un espace particulier autour du nid de la colonie. Pour cela, elles sélectionnent plusieurs sites de chasse et effectuent des explorations locales autour de ces sites. Il s'agit à la fois d'une stratégie globale et locale qui s'est révélée assez efficace pour maintenir jusqu'à nos jours l'existence de ces fourmis.

D'un point de vue biologique, ces fourmis vivent dans de petites colonies comportant rarement plus d'une centaine d'individus et établissent leur territoire d'exploration autour de leur nid en le partitionnant en sites de chasse. Chaque fourmi s'attribue des sites particuliers qu'elle va explorer aléatoirement en mémorisant toutefois préférentiellement ceux sur lesquels elle a pu rencontrer un certain succès. Au fur et à mesure de leur sortie, les ouvrières chargées de récolter de la nourriture s'éloignent de plus en plus du nid couvrant petit à petit une grande partie de leur espace de recherche. Plus généralement, trois règles décrivent le comportement de ces fourmis [Fresneau, 1994] :

- La découverte d'une proie entraîne toujours le retour sur le même site de chasse lors de la sortie suivante,
- La découverte d'une proie pèse sur la décision de sortie des fourrageuses en réduisant l'intervalle de temps passé au nid,
- Les fourrageuses semblent apprendre progressivement une association entre une direction de recherche opposée au nid et l'augmentation de la probabilité de succès.

D'autres contraintes pouvant avoir un impact dans un algorithme d'optimisation peuvent être mentionnées. Notamment, du fait de la fragilité du nid, des déménagements fréquents et réguliers ont lieu. Les sites choisis dépendent de la *qualité* des terrains découverts par des fourmis éclaireuses. On peut remarquer qu'un système de repérage par phéromone pourrait nuire à cette stratégie en *trompant* les fourmis incapables de distinguer des traces laissées menant à un nid maintenant abandonné. Afin d'éviter ce genre d'incident, les fourmis oublient toutes les connaissances qu'elles ont accumulées à chaque déplacement du nid.

Pour modéliser algorithmiquement ce comportement, deux opérateurs ont été définis afin de déterminer le déplacement et l'exploration des fourmis dans l'espace de recherche $\mathcal{S}$:

- l'opérateur $\mathcal{O}_{rand}$ qui génère un point de $\mathcal{S}$ de manière aléatoire et uniforme,
- l'opérateur $\mathcal{O}_{explo}(s, A)$ qui génère un point s' dans le voisinage d'un point s.

Ce dernier opérateur admet deux paramètres : le point de départ de l'exploration locale et l'amplitude maximale du voisinage à prendre en compte. Cet opérateur peut effectuer une simple exploration aléatoire ou être personnalisé avec une heuristique inspirée par le domaine de la recherche.

Pour adapter le comportement décrit ci-dessus, il est nécessaire d'examiner deux cas représentés sur la figure 2.3. D'un point de vue global, les fourmis explorent aléatoirement le voisinage d'un point central de l'espace de recherche représentant le nid des fourmis grâce à l'opérateur $\mathcal{O}_{rand}$. Elles cherchent des points de bonne qualité - maximisant la fonction d'évaluation - en mémorisant des sites de chasse localisés dans une

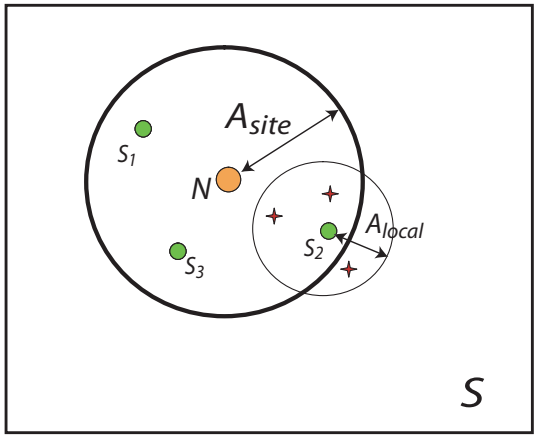


FIG. 2.3 – Exploration locale de la fourmi autour de son nid de rattachement. Trois sites de chasses s_i sont identifiés autour du nid N. La fourmi effectue une exploration locale depuis le site s_2.

amplitude A_{site} déterminée autour du nid N à l'aide de l'opérateur $\mathcal{O}_{explo}(N, A_{site})$. Au bout d'un temps donné, le nid est déplacé à un autre endroit et les fourmis oublient les sites de chasse mémorisés. D'un point de vue local, chaque fourmi se déplace autour du nid et mémorise p sites de chasse indépendamment des autres fourmis. Le voisinage de chaque site est exploré de manière aléatoire sur une zone délimitée par une certaine amplitude A_{local} (avec généralement $A_{local} < A_{site}$) en utilisant l'opérateur $\mathcal{O}_{explo}(s_i, A_{local})$ où s_i représente le point de départ de l'exploration. La capture d'une proie est représentée par une amélioration locale de la fonction d'optimisation. Lorsqu'une exploration mène à la capture d'une proie, la fourmi va systématiquement visiter le voisinage de ce site à la prochaine exploration. Et de manière symétrique, si un site se révèle peu fructueux, il est oublié et remplacé au sein de la mémoire par un nouveau site de chasse choisi aléatoirement dans le voisinage du nid.

L'automate présenté sur la figure 2.4 illustre le comportement individuel d'une fourmi pendant une recherche de nourriture. Comme indiqué dans le paragraphe précédent, une fourmi se crée initialement p sites de chasse en mémoire et décide d'en explorer un. À partir de ce site, la fourmi cherche des proies dans une amplitude A_{local} autour du site de chasse s_j. À chaque exploration, si une proie n'est pas trouvée - i.e. si la fonction d'optimisation n'est pas améliorée -, un compteur d'échec e_j est incrémenté et est réinitialisé dans le cas contraire. On considère que chaque fourmi peut effectuer un certain nombre de tentatives de chasses infructueuses avant d'abandonner son site de chasse. Ainsi, elles possèdent un paramètre P_{local} indiquant le nombre maximum d'échecs avant l'oubli du site de chasse.

À chaque itération de l'algorithme, toutes les fourmis du nid sont simulées en parallèle de manière indépendante. Ainsi, cet algorithme peut facilement être implémenté sur un système multiprocesseur afin d'établir une indépendance supplémentaire

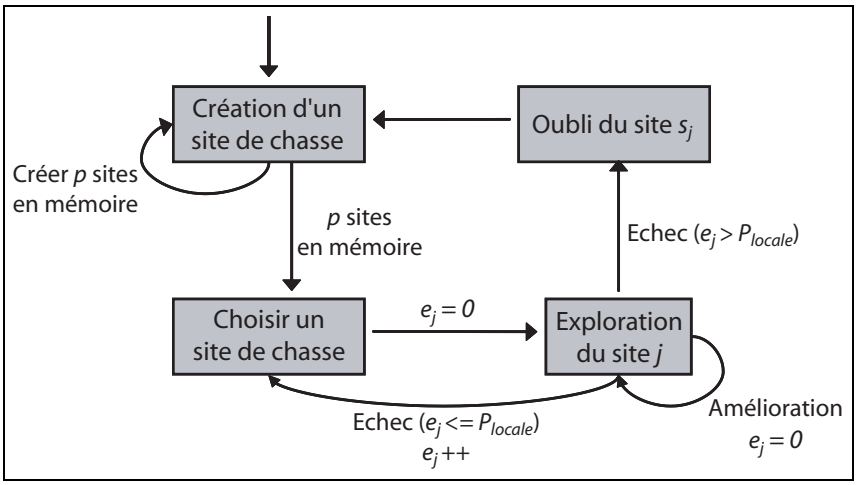


FIG. 2.4 – Automate représentant le comportement individuel de fourragement d'une fourmi.

entre chaque individu. Quelques extensions à cet algorithme ont été proposées dans [Monmarché, 2000]. Par exemple, une fourmi pourrait recruter un autre individu de la colonie si celui-ci explore une zone très fructueuse, augmentant ainsi les chances d'obtenir rapidement l'optimum recherché. Ou encore les paramètres pilotant le comportement des fourmis pourraient être fixés indépendamment pour chaque individu.

Les algorithmes inspirés par les colonies de fourmis sont des algorithmes à base d'agents particuliers. Cependant, il n'existe pas à notre connaissance d'outil de recherche d'information basé sur le comportement des fourmis. Nous nous inspirerons très fortement de cet algorithme afin de concevoir un modèle algorithmique relativement simple de recherche d'information sur Internet par algorithme de fourmis dans le chapitre 6.

Dans la suite, nous allons présenter des systèmes de recherche d'information conçus à partir d'algorithmes biomimétiques. Ces algorithmes ont montré leur grande adaptabilité dans beaucoup de domaines de recherche et il est donc naturel de penser à eux pour la résolution du problème que nous traitons ici.

2.4 Approches génétiques et à base d'agents pour le Web

Quelques auteurs se sont penchés sur la résolution de problèmes liés à Internet et plus particulièrement au problème de recherche d'information par algorithmes biomimétiques. On peut citer [Sheth, 1994] et [Morgan et Kilgour, 1996] utilisant l'apprentissage grâce à un AG, [Monmarché *et al.*, 1999a] utilisant un AG interactif afin de générer des styles de sites Web ou encore [Vakali et Manolopoulos, 1999] dont l'AG gère un cache permettant d'accélérer l'accès à des documents provenant de sources diverses. Dans les sections suivantes, nous allons détailler deux approches génétiques et une approche multi-agents apportant des solutions intéressantes à cette problématique.

Dans la littérature, on trouve facilement des algorithmes multi-agents traitant de ce problème, par exemple [Moukas, 1997] s'adapte à l'utilisateur en analysant les liens hypertextes qu'il parcourt.

2.4.1 Deux approches génétiques pour le problème de RI

Il existe plusieurs types de systèmes de recherche d'information sur Internet. Mais généralement, seuls les méta-moteurs de recherche utilisent des algorithmes évolutionnaires afin d'améliorer les résultats produits par une méthode plus classique à base d'index inversé. Les méta-moteurs ont en effet de plus grandes ressources matérielles à disposition pour effectuer une recherche parce qu'ils se présentent généralement comme une application mono-utilisateur et ne doivent par conséquent pas répondre à des milliers d'utilisateurs en parallèle. Sur ce principe, des agents intelligents ont vu le jour, agissant comme des assistants personnels de recherche en étant capable de s'adapter à l'utilisateur les manipulant.

2.4.1.1 Webnaut

Webnaut, décrit dans [Nick et Themis, 2001], est un bon représentant de ce type d'agents personnels. Il combine un méta-moteur interrogeant les moteurs de recherche classiques (*Google, ...*) avec un algorithme génétique permettant de générer de nouvelles requêtes afin de trouver de meilleurs résultats, le tout étant contrôlé par l'utilisateur au travers d'un système de *relevance feedback*. Cinq étapes se succèdent ainsi afin d'obtenir les meilleurs résultats possibles :

- Collecter des données sur les préférences de l'utilisateur (pages de résultats visitées du méta-moteur),
- Extraire des informations de ces données pour créer un profil de recherche (mots les plus représentatifs des documents),
- Découvrir de nouveaux documents en accord avec le profil créé précédemment,
- Évaluer et ne garder que les meilleurs documents découverts,
- Demander à l'utilisateur d'indiquer les meilleurs résultats retournés.

La première étape consiste donc classiquement à demander à l'utilisateur de poser une question au système de recherche à l'aide de mots-clés. Le méta-moteur interroge plusieurs moteurs de recherche et en extrait les résultats, élimine les doublons, et les présente à l'utilisateur sous forme de résumés. Ce dernier peut alors visiter les pages lui paraissant convenir le mieux à sa question. Pendant ce temps, le système enregistre les pages visitées et les propose en entrée du système génétique afin de créer de nouvelles requêtes. Le système génétique se compose de deux populations d'individus distincts représentés par des vecteurs de dimensions égales : une composée de termes et une composée d'éléments de la logique booléenne (et, ou, ...). Chacune est gérée et modifiée par deux algorithmes génétiques distincts l'un de l'autre. Les requêtes booléennes sont alors générées en combinant un individu de chaque population comme décrit ci-dessous.

Pour cela, les documents pris en compte sont représentés sous forme vectorielle - décrite dans la section 1.1.3 - dans laquelle les poids des mots sont calculés à partir

de leur fréquence d'apparition et de la fréquence du mot le plus représenté dans le document. La méthode classique *tf*idf* n'est pas utilisée ici parce qu'elle nécessite la prise en compte de l'ensemble des documents du corpus utilisé. Les individus de la première population de l'algorithme génétique sont alors représentés par une sélection aléatoire d'un nombre défini de termes, choisis parmi les mots les plus fréquents des documents concernés. Tandis que les individus de la seconde population sont générés aléatoirement à partir d'éléments de la logique booléenne.

Deux opérateurs génétiques sont utilisés dans chaque AG : un opérateur de croisement à un point de coupure agissant sur deux individus, et un opérateur de mutation. Cet opérateur consiste à choisir aléatoirement un élément d'un vecteur représentant un individu et à le remplacer par un autre, tiré au hasard dans le cas de la population de vecteurs de booléens, et issu des documents initiaux dans l'autre population.

Les requêtes sont alors générées en associant éléments à éléments les termes aux booléens. Elles sont ensuite triées comme indiqué sur l'illustration 2.5 puis évaluées. Cette évaluation consiste à interroger le méta-moteur grâce aux requêtes générées et à mesurer la similarité - par la mesure *cosine* - des nouveaux documents retournés avec ceux ayant servi initialement au système génétique.

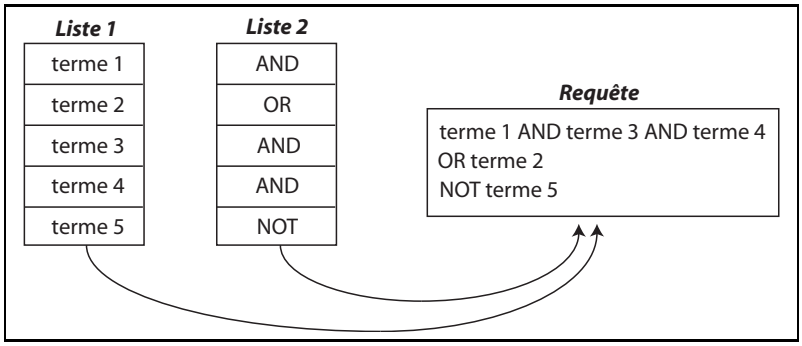


FIG. 2.5 – Génération d'une requête à partir de deux populations d'individus (une de termes et une d'éléments de la logique booléenne) dans le système *Webnaut*.

Les résultats produits par la meilleure requête sont alors présentés à l'utilisateur afin que celui-ci sélectionne les meilleures pages. Le processus peut alors recommencer dans le but d'améliorer encore les résultats obtenus.

2.4.1.2 Apprentissage par AG dans un agent

La capacité d'apprentissage des algorithmes génétiques a également été mise à contribution dans [Vallin et Coello, 2003]. Le but est également de créer un agent chargé d'assister les utilisateurs dans leurs problèmes de recherche d'information. Cependant, ce système a la particularité d'être conçu à la manière d'un outil de veille stratégique : il est chargé de scruter en permanence une zone prédéfinie d'Internet afin d'y détecter l'apparition de documents pertinents pour un utilisateur. L'originalité de ce système réside dans sa manière de récupérer les souhaits de l'utilisateur et de les adapter au fur et à mesure de son utilisation.

Cet agent accepte des requêtes sous forme de documents exemples dont le contenu doit être représentatif du souhait de l'utilisateur. Le système va ainsi scruter un site donné - la zone d'Internet prise en compte dans la recherche et indiquée par l'utilisateur - afin de détecter les pages les plus similaires aux exemples fournis en entrée à l'agent de recherche.

Afin de mesurer la similarité d'une page à une autre, les auteurs utilisent le modèle vectoriel de représentation des documents. Une sorte de liste noire de termes est utilisée afin d'éliminer les mots les plus courants du langage ainsi que ceux considérés comme trop communs dans le domaine de recherche considéré (comme par exemple le terme *'law'* pour une recherche portant sur un site gouvernemental). Les poids associés à chaque terme sont alors déterminés par la mesure *tf*idf*. Le corpus pris en compte pour le calcul de la composante *idf* correspond à l'ensemble des documents indiqués en entrée par les utilisateurs. Cet ensemble est alors désigné par le terme *requête*. Par contre, au cours de la recherche, il n'est pas possible de connaître l'ensemble du corpus pris en compte. Par conséquent, le calcul du vecteur représentatif d'un document en cours d'analyse ne pourra prendre en compte la composante *idf* et les poids des termes seront égaux à la valeur *tf*. La similarité peut alors être calculée grâce à la mesure *cosine* en utilisant les vecteurs de documents déterminés.

À partir de cette requête, l'agent analyse périodiquement l'ensemble des documents contenus dans le site visé. Lorsque des pages similaires au corpus représenté par la requête sont déterminées, elles sont présentées à l'utilisateur afin que ce dernier puisse établir une note. La requête initiale composée de documents choisis par l'utilisateur peut ainsi évoluer afin de prendre en compte les modifications de souhaits des utilisateurs ou la découverte de pages plus intéressantes au cours de la recherche. Cette adaptation, sorte d'apprentissage, se fait conjointement à l'aide d'un système de *relevance feedback* et d'un AG.

La population prise en compte dans l'AG est composée d'individus représentés par un vecteur de termes représentant un document correspondant à la requête. La fonction de fitness utilisée calcule une valeur représentant l'adaptation de cette requête à l'intérêt de l'utilisateur. Un opérateur de croisement à deux points est utilisé afin de *mélanger* les gènes correspondant aux termes représentés dans les vecteurs de documents, dimensionnés de manière similaire. L'opérateur de mutation consiste à sélectionner aléatoirement des poids de termes dans les vecteurs de documents et à les modifier en fixant une valeur choisie au hasard entre 0 et 1.

L'utilisateur peut renforcer ou atténuer l'intérêt d'un document en lui donnant un score respectivement positif ou négatif. Les poids des termes dans les vecteurs représentant les individus sont alors mis à jour en conséquence et la valeur de fitness calculée est augmentée ou diminuée de la même manière.

2.4.1.3 Autres travaux mettant en œuvre des algorithmes génétiques

Les deux applications présentées dans cette section concentrent l'effort d'optimisation des AG qu'ils utilisent à la modification de la représentation des documents et à l'adaptation de la représentation de la requête initiale formulée par l'utilisateur. Il existe

d'autres exemples de telles modélisations, citons par exemple [Sanchez et Pierre, 1994], [Kraft *et al.*, 1997] ou [Landrin-Schweitzer *et al.*, 2003]. Ce dernier exemple utilise un AG qui construit des requêtes d'interrogation de moteurs de recherche en les adaptant pour chaque utilisateur. L'utilisateur émet ensuite un avis sur la pertinence des requêtes formées afin que le système apprenne ses goûts pour les requêtes suivantes.

Mais d'autres applications des AG dans la recherche d'information existent. Par exemple [Pathak *et al.*, 2000] se sert de leurs propriétés afin d'adapter la fonction d'analyse des documents ou encore [Chen, 1995] utilise un AG afin d'optimiser les mots-clés à rechercher dans un document pour pouvoir l'évaluer correctement.

Dans la section suivante, nous présentons un algorithme à base d'agents capables d'évoluer grâce à un système d'apprentissage par renforcement. Ce modèle détermine une approche très intéressante qui démontre que l'utilisation de nouveaux algorithmes de recherche n'utilisant pas une indexation peuvent apporter une information plus approfondie et pertinente que les moteurs de recherche traditionnels.

2.4.2 Une approche multi-agents : *InfoSpiders*

Les moteurs de recherche classiques utilisent des *crawlers* afin de parcourir Internet dans le but d'indexer un maximum de pages. Ces *robots* agissent comme des agents relativement peu évolués. En effet, ils cherchent uniquement à parcourir l'ensemble d'Internet, sans stratégie particulière pour trouver une page intéressante pour une requête donnée. L'effort d'indexation est rentabilisé dans les moteurs de recherche par la très grande réutilisation de cet index afin de répondre aux requêtes des utilisateurs.

Cependant, à la vue des résultats de l'analyse des caractéristiques d'Internet (voir section 1.2.1) certains auteurs ont émis l'hypothèse que des agents *intelligents*, qui arriveraient à déterminer efficacement les liens hypertextes utiles et intéressants d'une page, peuvent trouver relativement rapidement des documents pertinents à une requête donnée [Albert *et al.*, 1999].

En partant de cette hypothèse, Menczer a conçu le modèle *InfoSpiders* décrit initialement dans [Menczer et Monge, 1999] puis [Menczer, 2003]. Ce système est composé de plusieurs agents évoluant de leur *naissance* à leur *mort* et chargés de retrouver en permanence les documents correspondant à une requête donnée. Un tel système, au contraire d'une approche classique à base d'index, a l'avantage de prendre en compte naturellement le caractère de plus en plus dynamique des documents du réseau évoluant de jour en jour.

L'utilisation d'agents complètement autonomes décompose le problème général en plusieurs sous-problèmes que chaque entité va se charger de résoudre. Cette stratégie est inspirée de l'adage *«diviser pour régner»*. Chaque individu va ainsi naviguer de documents en documents en prenant lui-même ses décisions sur les liens à suivre dans le but de trouver des pages pertinentes à une requête formulée par un utilisateur. Un agent va être vu comme le navigateur personnel d'un utilisateur qui va chercher les informations voulues en adaptant sa stratégie en fonction du contexte local et des préférences de

l'utilisateur. L'apprentissage qui en découle est réalisé de manière individuelle par la technique de l'apprentissage par renforcement. La spécialisation des agents est assurée par le fait qu'aucune information ne peut être communiquée d'agent à agent et qu'un individu ne peut percevoir d'information globale sur l'espace de recherche. Ainsi, seules les interactions locales agissent dans le processus d'apprentissage. En agissant de cette manière, on maintient une diversité qui permet de couvrir un maximum d'aspects liés à la requête.

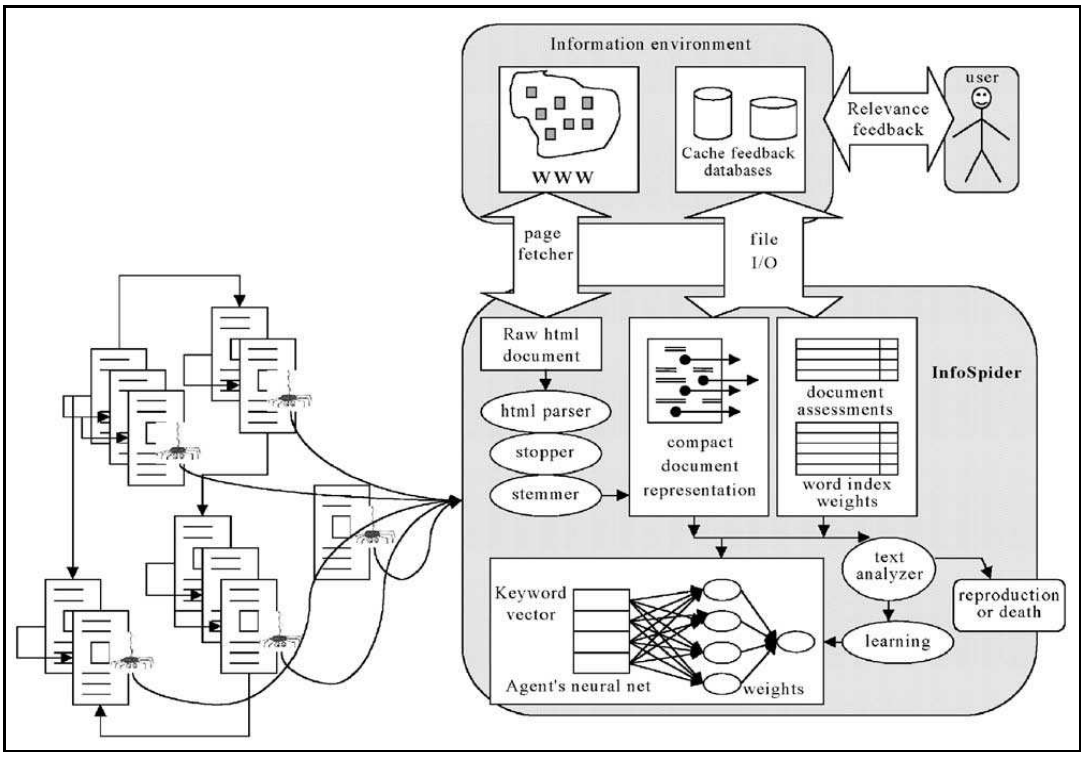


FIG. 2.6 – Architecture d'un agent dans le modèle *InfoSpiders*.

L'architecture d'un agent dans le système *InfoSpiders* est représentée par la figure 2.6. Afin de pouvoir évaluer chaque agent de manière à sélectionner efficacement les plus performants d'entre eux, tous les individus possèdent une certaine énergie fixée uniformément à l'initialisation. Chaque agent est alors placé à l'origine sur une page considérée comme un bon élément de départ, une page relativement pertinente par rapport à une requête donnée. Afin de garantir de *bons* points de départ, ceux-ci sont issus de moteurs de recherche classiques, interrogés à l'aide de la requête de l'utilisateur, ou de signets définis par l'utilisateur lui-même.

On affecte à chaque agent un réseau de neurones dont les entrées correspondent à la liste des mots-clés formulés par l'utilisateur dans la requête. À chaque itération - dès qu'un agent se retrouve sur un nouveau document - les araignées chercheuses analysent le document qui leur est attribué afin de déterminer les meilleurs liens à suivre par la suite. Cette analyse consiste à examiner la présence des mots-clés de la requête dans le

voisinage d'un lien. Le poids de chaque entrée du réseau de neurones - correspondant à un mot-clé - est mis à jour en fonction de son éloignement à un lien. Ce score est d'autant plus élevé que le mot-clé est proche d'un lien. Le réseau de neurones fait ensuite la somme de toutes les activités de ses entrées afin de calculer une fonction d'activation pour chaque lien. Le lien hypertexte que l'agent va par la suite suivre est sélectionné aléatoirement en favorisant les liens ayant obtenu le meilleur score au travers du réseau de neurones.

L'énergie de chaque agent est ensuite mise à jour en fonction de la pertinence à la requête du document analysé et du coût de téléchargement de la nouvelle page à visiter. Ces éléments étant à définir pour chaque implémentation du modèle. Le réseau de neurones est également mis à jour en fonction de la pertinence de la page analysée et de l'estimation que le réseau en avait faite à l'itération précédente en analysant le lien que l'agent a suivi. Cette adaptation est réalisée à l'aide d'une version adaptée à un réseau de neurones de l'algorithme *Q-Learning* [Watkins, 1989, Watkins et Dayan, 1992, Lin, 1992] réalisant un apprentissage par renforcement. Les poids utilisés dans le réseau de neurones sont alors mis à jour par rétro-propagation de l'erreur.

Si l'énergie d'un agent est trop faible, il est détruit. Dans le cas inverse, lorsque l'énergie d'un agent a atteint un certain seuil, celui-ci va se dupliquer en apportant un bruit dans les poids du réseau de neurones du descendant et en ajoutant dans la liste des mots-clés utilisés ceux apparaissant le plus fréquemment dans le document actuel. Cette modification s'apparente à une mutation comme on la retrouve dans un AG et permet à une population d'agents de s'adapter à l'environnement et d'explorer des zones potentiellement intéressantes.

Dans [Pant et Menczer, 2002], les auteurs ont réalisé une implémentation en Java du modèle *InfoSpiders* appelée *MySpiders* disponible sur Internet. Cette application illustre le potentiel de ce système. Les agents sont implémentés en plusieurs processus s'exécutant de manière concurrentielle. De cette manière, leur indépendance est encore plus forte.

Afin de déterminer le taux d'énergie apporté en récompense de la découverte d'une page pertinente, la mesure de similarité *cosine* est utilisée entre un vecteur correspondant aux termes de la requête et un autre vecteur correspondant aux termes du document analysé. Les poids utilisés dans le modèle vectoriel ne comprennent que les fréquences des documents - la composante *tf* dans le modèle *tf*idf* - car l'agent ne peut obtenir aucune information globale sur l'espace de recherche qu'il explore. Le coût de téléchargement est, quant à lui, lié à la latence observée dans le protocole d'échange de fichier sur Internet : le protocole *HTTP*. Ainsi les agents explorant des pages se situant sur des serveurs très lents ou saturés auront tendance à disparaître plus rapidement que les autres.

Enfin, l'évaluation de ce système a montré qu'il est très difficile de comparer un tel outil aux moteurs de recherche standards. En effet, il s'agit de deux approches complémentaires mais très différentes. Mais toutefois il semble que ce système d'adaptation de la requête mène à trouver des pages plus pertinentes et plus récentes que dans l'approche traditionnelle.

2.5 Conclusion

Ce chapitre nous a permis d'aborder des heuristiques de recherche d'information sur Internet. Nous montrerons dans les chapitre suivants que les approches biomimétiques (algorithmes génétiques et à base de population de fourmis) maintenant utilisés classiquement dans le domaine de l'optimisation s'adaptent particulièrement bien à ce type de problème. La méthode tabou est elle-aussi très utilisée en optimisation et a montré des qualités pour la résolution de multiples problèmes.

Les propriétés de parallélisation de chacune des méthodes nous permettra, dans la suite de cette thèse, de réaliser un outil efficace de recherche d'information sur Internet. Les ressources à la disposition de notre système de recherche seront alors exploitées avec tout leur potentiel.

À partir de ces éléments, nous avons décidé d'élaborer un outil de recherche complémentaire aux moteurs de recherche classiques. Après avoir présenté une modélisation du problème nous permettant de le transformer en un problème d'optimisation (chapitre 3), nous définirons dans les chapitres suivants des approches inspirées de la littérature de résolution de problèmes d'optimisation et plus particulièrement des algorithmes évolutionnaires.

Chapitre 3

Modèle générique de recherche d'information sur Internet

Résumé

Dans ce chapitre, nous allons définir un modèle générique de recherche d'information sur Internet. Cette approche introduit une notion complémentaire aux moteurs de recherche classiques. On considère que l'utilisateur peut attendre un certain temps les résultats de sa recherche afin de laisser l'outil mis en place analyser plus profondément les résultats grâce à une requête utilisateur très détaillée. Ce modèle code le problème de RI sous un forme générique autorisant l'application de tout type de méta-heuristique d'optimisation afin de résoudre le problème initial. Il offre également une architecture permettant de réaliser une application distribuée dans laquelle plusieurs ordinateurs clients peuvent trouver une place et participer à la recherche globale.

3.1 Codage du problème de RI dans notre modèle

Nous proposons un modèle pour un moteur de recherche qui peut être considéré non pas comme concurrent des moteurs classiques mais plutôt comme une approche complémentaire. Dans cette approche, nous considérons qu'un utilisateur est prêt à attendre son résultat pendant plusieurs heures au profit d'un gain de qualité. Le temps disponible est alors mis à contribution afin d'analyser individuellement et précisément toutes les pages souhaitées, permettant d'obtenir une évaluation plus pertinente de leur contenu en fonction d'une requête utilisateur complexe.

En effet, si l'on considère que la machine dispose de quelques heures pour donner un résultat, on peut envisager d'effectuer des traitements supplémentaires qui font défaut aux moteurs classiques. Il s'agit en particulier de formuler une requête plus riche, de télécharger les pages pour obtenir une version actualisée et mieux analysée en terme de

contenu - comme le préconise Lawrence dans [Lawrence et Giles, 1999] - de proposer une classification des pages trouvées en fonction de leur contenu textuel - comme décrit dans la section 1.4.2.2 - ou d'effectuer une recherche avec une stratégie donnée.

Ce modèle permet à un outil de recherche d'aider l'utilisateur dans sa tâche d'analyse des résultats et de tenter de raccourcir le temps mis à disposition. Il peut également servir de base à un outil de veille stratégique fournissant à intervalles réguliers un rapport complet sur la manière dont est traité un sujet particulier sur Internet.

L'espace de recherche utilisé dans un moteur de recherche est Internet dans sa globalité. Pour transformer notre problème de recherche d'information en un problème d'optimisation, nous avons modélisé le Web comme un graphe dont les nœuds sont les pages, et les arcs, les liens hypertextes qui existent entre les pages comme illustré sur le schéma 3.1. Cette modélisation est utilisée dans de nombreuses problématiques liées à Internet [Albert *et al.*, 1999, Broder *et al.*, 2000] et permet de déterminer certaines caractéristiques du réseau Internet. Nous avons abordé ce sujet dans la section 1.2.1 du chapitre 1.

En choisissant une représentation de ce type, il est à la fois possible d'utiliser des algorithmes à base d'agents parcourant des routes de documents en documents comme on a pu le rencontrer dans la section 2.4.2 avec le modèle *InfoSpiders*, des algorithmes issus de la théorie des graphes, ou enfin des algorithmes utilisés dans le domaine de l'optimisation. Cette dernière catégorie nécessite toutefois de préciser certains points afin de déterminer un espace de recherche mathématique.

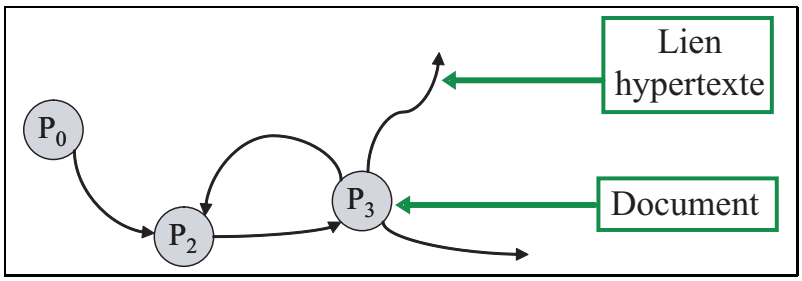


FIG. 3.1 – Modélisation d'Internet sous forme de graphe. Les documents représentent les nœuds du graphe et les liens hypertextes contenus dans les documents les arcs orientés de ce même graphe.

Nous prenons en compte le graphe obtenu comme un espace de recherche que l'on va noter S dans la suite de cette thèse. Cette modélisation nous permet de considérer que chaque nœud du graphe correspond à un point de S. La notion d'arc du graphe est liée aux liens hypertextes des documents. Elle établit une relation de voisinage pour chaque point de cet espace de recherche. Il est également possible d'attribuer une valuation à chaque nœud en définissant une fonction d'évaluation dont les paramètres correspondent à la requête fournie par l'utilisateur au moteur de recherche. Chaque point de S peut alors être évalué et possède une relation de voisinage.

L'espace originel que représente Internet est alors transformé en un espace de re-

cherche mathématique et le problème de RI consiste alors à trouver les optima de cet espace de recherche. Chaque requête de l'utilisateur va représenter une fonction d'évaluation différente et un paysage de qualité différent. Avec ce formalisme, il devient possible d'appliquer des algorithmes d'optimisation de fonctions pour obtenir les solutions (les pages Web) à la problématique d'un moteur de recherche.

La modélisation que nous utilisons propose donc un parallèle entre le problème de RI sur le Web et le problème général d'optimisation d'une fonction qui peut être résumé par le tableau 3.1.

Optimisation	**Internet**
Espace de recherche	Ensemble des pages
Fonction d'évaluation	Adéquation de la page à la requête utilisateur
Solution optimale	Page maximisant l'adéquation à la requête
Relation de voisinage	Définie par les liens sortant d'une page

TAB. 3.1 – Modélisation du problème de la recherche d'information sur le Web comme un problème d'optimisation

Cet espace de recherche étant bien particulier, il est nécessaire de définir des opérateurs permettant de l'explorer ou plus généralement de le manipuler. La section suivante est dédiée à la définition de tels opérateurs adaptés à notre modèle. Dans le contexte décrit précédemment, il est également nécessaire de définir une fonction d'évaluation des points - donc des documents - de S. La section 3.3 présente deux évaluations liées à deux types de requêtes utilisateurs que nous avons définies dans le contexte de la création d'un méta moteur de recherche implémentant notre modèle.

3.2 Opérateurs de recherche

Pour parcourir l'espace de recherche S, les algorithmes d'optimisation utilisent des opérateurs de recherche. Il en est de même pour les moteurs de recherche. On peut cependant distinguer plusieurs types d'opérateurs permettant d'explorer de manière relativement différente cet espace de recherche.

Dans la suite de cette section, nous allons définir deux opérateurs génériques qu'il est possible d'utiliser dans tous les algorithmes étudiés dans cette thèse. Le premier O_{rand} génère un nouveau point dans l'espace de recherche indépendamment de tout autre point. Cet opérateur est au moins nécessaire afin d'obtenir une sorte de point d'entrée dans notre espace de recherche. Le deuxième, O_{explo}, permet d'examiner le voisinage local d'un point particulier de S. C'est un opérateur bien connu et utilisé dans de très nombreuses heuristiques en optimisation.

3.2.1 Opérateur de création heuristique O_{rand}

Ce type d'opérateur est utilisé afin de créer des points de S sans tenir compte d'aucune information issue d'autres points. Cet opérateur est similaire aux opérateurs

suivants utilisés en optimisation : génération aléatoire d'un point dans l'espace de recherche, ou bien création heuristique d'un point. Cependant, dans le cas du Web, il est difficile d'adapter une telle fonction de hasard. En effet, même si on considère l'espace de recherche comme un graphe, il est nécessaire de pouvoir situer un point par des coordonnées. Or, sur Internet, cette localisation se fait par une URL composée d'une adresse - une adresse IP ou un nom de domaine correspondant à une adresse IP - identifiant un serveur Web sur le réseau et d'un chemin identifiant un document particulier sur ce serveur. Cependant, l'ensemble formé par toutes les adresses IP disponibles inclut l'ensemble des adresses IP utilisées par un serveur Web sur Internet. En effet, une adresse IP est formée de 4 octets, il existe donc 256^4 soit un peu plus de 4 milliards de possibilités, et à l'heure actuelle seulement 50 millions de serveurs sont recensés.

Générer aléatoirement des adresses IP a déjà été étudié [Lawrence et Giles, 1999, Kishi *et al.*, 2000] mais ne donne une adresse valide qu'avec très peu de chance, et une adresse de site Web avec une probabilité encore plus faible. En effet, sur les expérimentations menées par Lawrence en 1999, seulement une adresse sur 269 semblait valide et seulement $\frac{1}{6}$ de ces serveurs présentaient des pages accessibles directement. Donc ce type d'opérateur aléatoire ne semble pas être utilisable pour explorer efficacement le Web.

Un autre exemple d'opérateur de création dans les problèmes d'optimisation consiste à utiliser une heuristique qui va construire une solution en fonction des données du problème. Dans le cas du Web, de nombreux moteurs de recherche utilisent un tel opérateur qui consiste à interroger l'index des moteurs classiques et à donner en sortie le ou les liens fournis par ces moteurs. C'est le cas notamment des méta-moteurs qui compilent des résultats - par exemple [Sander-Beuermann et Schomburg, 1998] - ou des agents - voir section 2.4.2 sur le système *InfoSpiders* - qui initialisent leur recherche avec ces résultats.

Nous allons définir l'opérateur O_{rand} sur ce modèle. O_{rand} est conçu de manière à générer un nouveau point dans notre espace de recherche en prenant un résultat issu d'un moteur de recherche existant. Ces résultats sont pris dans l'ordre original donné par les moteurs de recherche en alternant les demandes sur chacun d'eux. Ainsi, le premier document retourné correspondra au premier lien du premier moteur, puis le premier lien du deuxième et ainsi de suite. La requête utilisateur mise en œuvre est adaptée, généralement sous forme d'une liste de mots-clés, afin de pouvoir interroger des moteurs classiques et obtenir des résultats. Les moteurs utilisés actuellement dans notre application sont *Google*, *Altavista*, *Lycos*, *Teoma* et *Yahoo*. Il faut noter que les moteurs limitent généralement le nombre de réponses pour une requête (1000 par exemple pour *Google*) et que par conséquent il s'agit d'une ressource relativement limitée surtout dans le cadre d'une application de veille stratégique.

3.2.2 Opérateur d'exploration locale O_{explo}

Ce deuxième type d'opérateur que l'on peut également appeler opérateur de transformation va se servir de points existants pour en générer de nouveaux. Les techniques

à base d'agents utilisent généralement ce type de stratégie en explorant les liens trouvés dans les différents documents rencontrés. Ce type d'opérateur est également largement utilisé en optimisation : mutation dans un algorithme génétique, exploration locale dans un algorithme glouton, etc. Ainsi, un opérateur classique en optimisation comme l'ascension locale peut être directement transposé sur le Web : à partir d'une page Web donnée, il est possible d'obtenir l'ensemble de ses voisins, de les explorer un par un et de choisir le meilleur, selon une fonction d'évaluation donnée, comme nouveau point de départ.

De cette manière, nous définissons l'opérateur $O_{explo}(P, S_V)$, $S_V \geq 1$ permettant d'explorer le voisinage d'un point P de S. Dans le cas du Web, le voisinage est formé par les pages pointées par les liens hypertextes. Ainsi, en parcourant les liens hypertextes issus de la page P prise en compte, on obtient un ensemble de nouveaux points voisins de P dans S. La taille du voisinage S_V précisée en paramètre de cet opérateur correspond à la profondeur des liens explorés.

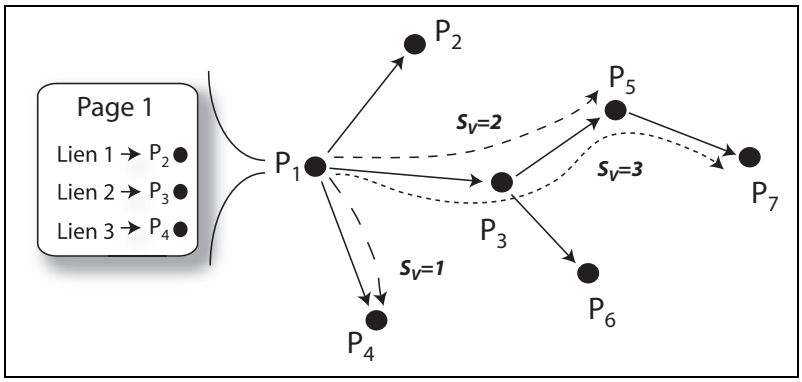


FIG. 3.2 – Modélisation de la taille du voisinage S_V utilisé dans l'opérateur O_{explo}.

La figure 3.2 illustre la taille du voisinage S_V utilisée dans l'opérateur O_{explo}. À partir des pages Web composant un réseau, on effectue la transformation sous forme de graphe présentée dans la section 3.1. Les pages P_2 à P_4 correspondent aux cibles des liens hypertextes contenus dans la page P_1 et sont dans le voisinage $S_V = 1$ de P_1. Les pages P_5 et P_6 sont dans un voisinage $S_V = 2$ de P_1 tandis que la page P_7 est dans un voisinage $S_V = 3$ de P_1.

Il est donc nécessaire, dans un premier temps, d'extraire et de lister les liens des documents considérés. Ensuite, l'opérateur donne en sortie un nouveau point qui peut être dans le voisinage immédiat de la page ($S_V = 1$) - c'est-à-dire les pages pointés par les liens trouvés dans la page P - ou bien plus loin dans le graphe des relations de voisinage. S'il n'est pas possible d'aller à la profondeur désirée - lorsque des documents ne comportent pas de liens -, l'opérateur retournera le point rencontré à la profondeur maximale. La politique de sélection des points à explorer à chaque niveau de profondeur est discutée dans les paragraphes suivants.

Nous avons vu dans la section 1.2.1 que le diamètre d'Internet est en moyenne de 21 liens. Par conséquent, ce type d'opérateur peut couvrir une grande zone de l'espace de recherche avec un voisinage relativement faible et, s'il est bien guidé, retrouver rapidement l'information voulue. Cela rejoint les constatations d'Albert, Jeong et barabási dans [Albert *et al.*, 1999] qui concluent qu'un agent intelligent suivant uniquement des liens hypertextes sur Internet peut retrouver relativement rapidement n'importe quelle information.

Une propriété importante de cet opérateur est la possibilité de sélectionner les liens à explorer en fonction du contexte l'entourant. Nous utilisons dans ce but une heuristique déjà mise à profit dans d'autres études : le texte servant d'ancrage aux liens ainsi que le texte placé autour des liens est une indication parfois fiable des informations que l'on trouvera dans la page pointée. Il est donc possible d'ordonner les liens d'un document, par exemple en fonction de la fréquence des mots-clés dans le texte entourant ces liens, ou plus généralement suivant la fonction d'évaluation construite autour de la requête de l'utilisateur. L'opérateur a alors la possibilité de choisir un lien au hasard, de sélectionner le premier de la liste ordonnée, un lien au hasard parmi les premiers ou selon une probabilité décroissante en fonction de la position dans la liste.

3.3 Fonctions d'évaluation et requête utilisateur

La modélisation mise en œuvre nécessite de pouvoir évaluer chaque nœud du graphe représentant notre espace de recherche. Il est ainsi nécessaire d'établir une fonction d'évaluation capable d'analyser un document et de fixer une valeur à chaque point de l'espace ou au moins une relation de dominance d'un point sur un autre.

La construction d'une fonction d'évaluation pour un moteur de recherche nécessite l'élaboration d'une requête utilisateur. Cette requête doit permettre de récupérer les souhaits de l'utilisateur du mieux possible afin de pouvoir déterminer avec précision les documents les plus pertinents au sens de l'utilisateur.

Dans la suite de cette section, je vais détailler deux fonctions d'évaluations utilisées dans cette thèse. La première fonction permet d'obtenir un score pour chaque point de l'espace de recherche indépendamment d'un autre. Cependant lorsque, pour une page donnée, un paramètre pris en compte dans l'évaluation domine très fortement les autres, la valeur obtenue a tendance à être trop importante vis-à-vis des autres pages. La deuxième fonction d'évaluation présentée tente de résoudre ce problème en utilisant un critère de dominance selon Pareto tout en proposant à l'utilisateur de choisir les critères à optimiser en priorité.

3.3.1 Première évaluation par mesures statistiques

L'évaluation d'une page P commence nécessairement par le téléchargement de la page (amputée des images, des scripts ou encore des différents objets qui lui sont liés) suivi d'une analyse syntaxique de la page qui permet d'extraire des informations comme le contenu textuel (avec ponctuation), les fichiers référencés (images, sons, etc), les

adresses email, les liens vers d'autres pages Web. La fonction d'évaluation f quantifiant la qualité d'une page P peut alors être formulée comme une fonction numérique de la requête de l'utilisateur. Chaque type de document rapatrié d'Internet va être converti en un format comprenant les liens hypertextes, le texte et les informations typographiques (mots en gras, italique, titres, ...). Nous ne détaillons pas plus ici les techniques qui permettent d'obtenir de manière fiable ces informations, mais il faut savoir que cela pose des problèmes (accessibilité des serveurs, syntaxe HTML erronée, etc).

Afin de pouvoir analyser les documents plus précisément que ne le font les moteurs de recherche classiques, nous avons défini une requête utilisateur plus riche que celles utilisées généralement. Classiquement, une requête utilisateur est composée d'une liste de mots-clés pouvant être reliés par des expressions booléennes. Cependant, la logique booléenne possède ses limites et ne permet pas de spécifier assez d'informations afin de *capturer* précisément le souhait de l'utilisateur. En effet, le langage naturel est beaucoup plus riche que ce que permet d'appréhender la logique booléenne, on ne peut par exemple pas exprimer une notion d'adjacence de mots-clés ou de répartition de mots-clés dans un document dans le cadre de la logique booléene. Il faut par conséquent beaucoup d'expérience à un utilisateur pour convertir une pensée du langage naturel vers cette logique. Et sans cette expérience il est difficile de cerner exactement ce que désire l'utilisateur.

Dans cette optique, nous avons défini une requête comprenant un plus grand nombre de critères comme par exemple la présence impérative ou non de mots-clés - que nous dénoterons K_i dans la suite de ce document -, le nombre d'occurrence de ces mots-clés, leur répartition dans le texte - autrement dit leur proximité à des liens hypertextes ou à d'autres mots-clés -, la présence souhaitée de mots - mots notés S_i dans la suite -, l'absence souhaitée de mots - notés Sn_i. Chacun de ces critères peut être déterminé par une analyse statistique de la présence des mots-clés dans le texte des pages Web analysées.

De cette manière, la requête utilisateur se compose de plusieurs listes de mots-clés ayant chacune une signification particulière :

- la première rassemble l'ensemble des K_i. Ces éléments vont être utilisés par l'opérateur O_{rand} afin de construire une requête compréhensible pour les moteurs de recherche classiques ;
- Les mots *souhaités* par l'utilisateur sont rassemblés sous la liste des S_i. Ces termes ne sont pas considérés avec autant d'importance que les K_i dans la valeur déterminée par la fonction d'évaluation ;
- L'utilisateur peut indiquer ici les mots qu'il *préférerait* ne pas voir figurer dans les documents retournés en résultats dans la liste des Sn_i. La présence d'un Sn_i dans une page Web va alors réduire la valeur obtenue par la fonction d'évaluation. On peut toutefois noter qu'il n'est pas interdit de voir figurer un tel mot dans un résultat ;
- l'utilisateur peut également indiquer à la fonction d'évaluation les termes qui doivent *obligatoirement* être présents dans les documents. Pour cela, il dispose d'une quatrième liste : les M_i ;

– à l'inverse, il peut être nécessaire de spécifier certains mots à ne pas retrouver dans un document. C'est le but de la dernière liste de mots : les Mn_i.

De cette manière, si l'on souhaite voir apparaître des résultats ayant trait aux algorithmes évolutionnaires, mais ne portant pas sur les AG et traitant de préférence sur des algorithmes à base de fourmis artificielles et non à base de nuages d'insectes, on pourra formuler la requête de cette manière :

– K_i : {algorithme, évolutionnaire}
– S_i : {fourmis, artificielles}
– Sn_i : {nuages, insectes}
– M_i : {évolutionnaire}
– Mn_i : {génétique, AG}

On notera que dans l'exemple précédent, le mot «évolutionnaire» est présent à la fois dans les K_i et dans les M_i. Ce n'est pas une obligation, mais une possibilité, qui a pour effet d'accentuer le poids de ce terme dans la fonction d'évaluation et d'imposer sa présence.

L'analyse des liens est très importante dans la modélisation que nous avons adoptée, notamment pour permettre une bonne exploration locale (voir section 3.2.2). Cette analyse se fait elle aussi en fonction de la requête utilisateur en analysant les mots-clés spécifiés dans le voisinage de chaque lien. Nous analysons le texte qui a servi d'ancrage au lien ainsi que le texte placé autour du lien. En effet, cette méthode se base sur le fait qu'un lien a plus de chances d'aboutir vers une page intéressante si elle est à proximité d'un ou plusieurs mots-clés. La taille du voisinage pris en compte dans l'analyse est très importante et doit idéalement être délimitée en fonction du contexte linguistique dans lequel apparaît le lien. Cependant, il n'existe pas à l'heure actuelle de méthode fiable pour effectuer cette opération. Nous avons donc choisi de tirer parti de la ponctuation apparaissant dans le texte entourant le lien. En effet, un lien figurant au début d'un paragraphe aura moins de chances de pointer vers une page dont les termes correspondent à ceux du paragraphe précédent. Le voisinage analysé correspond ainsi à une zone définie par les contraintes suivantes :

– les cinq premiers mots seront examinés dans tous les cas,
– la recherche s'arrête si un caractère de fin de phrase (point, paragraphe, point d'interrogation, ...) est rencontré,
– la zone examinée ne doit pas dépasser vingt mots.

Dans cet espace de recherche limité aux abords du lien, on détermine la présence des divers mots-clés exprimés dans la requête afin de calculer un score au lien. On attribue à chaque mot-clé détecté une valeur de base en fonction de son origine : un mot S_i aura une valeur inférieure à un mot K_i. Cette valeur de base va ensuite être pondérée par une fonction décroissante avec la distance au lien comme illustré dans la figure 3.3. La valeur associée au lien sera alors la somme du poids de chaque mot trouvé dans l'espace de recherche. On peut remarquer qu'un mot-clé peut *voter* pour plusieurs liens suivant sa position dans le texte.

La valeur obtenue par chaque lien rentre en compte dans l'évaluation d'un document. Nous augmentons de cette manière la fitness des pages qui contiennent des liens pointant

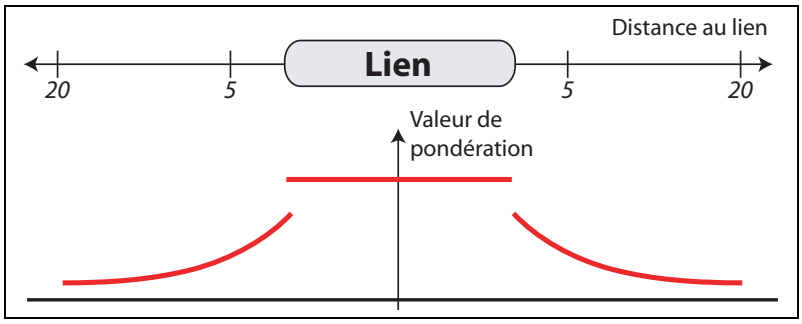


FIG. 3.3 – Attribution de poids dans l'évaluation d'un lien.

potentiellement vers des pages intéressantes. Cette disposition permet de rendre plus efficace l'exploration locale par l'opérateur O_{explo}.

Finalement, la formule 3.1 définit la manière dont est calculée la valeur de chaque page.

$$f(P) = \begin{cases} 0 \text{ si } \exists i / \#(M_i) = 0 \vee \exists i / \#(Mn_i) \neq 0 \\ Uniform(K_i) \times \left[\omega_K \sum_i \#(K_i) + \omega_S \sum_i \#(S_i) + \right. \\ \qquad \left. \omega_{Sn} \sum_i \#(Sn_i) + \omega_L \sum_i Link_{eval}(L_i) \right] \text{ sinon} \end{cases} \tag{3.1}$$

Dans cette équation, $\#(a)$ signifie nombre d'occurrences du mot a dans la page P. P obtient donc un score de 0 si un seul des mots-clés $(M_1, M_2, \dots)$ manque ou si un seul des mots $(Mn_1, Mn_2, \dots)$ est présent, et plus le nombre de mots-clés présents est élevé, plus le score de la page est élevé. $Link_{eval}(L_i)$ est une fonction qui évalue l'intérêt du i-ème lien trouvé dans P comme décrit dans le paragraphe précédent.

La fonction $Link_{Eval}$ pour un lien L donné est d'autant plus élevée que des mots-clés de $(K_1, K_2, \dots)$ sont présents dans le voisinage textuel du lien. Les poids ω_K, ω_S, ω_{Sn} et ω_L servent à doser l'importance relative des différents composants de la requête.

Enfin, $Uniform()$ est une fonction qui favorise une représentation équitable des mots-clés et prend une valeur maximale de 1 lorsque les proportions d'apparition des mots-clés dans la page P sont toutes égales. Cette fonction sert à récompenser les pages dans lesquelles aucun mot-clé ne domine les autres. Elle assure ainsi que le domaine thématique de la recherche défini par l'ensemble des mots-clés est bien respecté et qu'un mot-clé ne viendra pas *dominer* les autres en proportion. Elle est fondée sur une notion proche de l'entropie - représentée par l'équation 3.2 - qui ne diffère d'elle que dans les cas où l'entropie prend une valeur trop proche de 0. En effet, dans cette situation, la page serait condamnée à ne pas figurer dans les résultats. C'est le cas par exemple d'un document dans lequel un mot-clé est extrêmement plus représenté que les autres mots-clés.

$$H = -\frac{\sum_{i=1}^{n} p_i \cdot ln(p_i)}{ln(n)} \tag{3.2}$$

Où n désigne le nombre d'éléments pris en compte et p_i désigne la proportion du $i^{ème}$ élément par rapport à tous les autres.

Autrement dit, $f(P)$ sera élevée si P contient des mots-clés en proportions égales, le plus de mots possible de $(S_1, S_2, \dots)$, le moins de mots possible de $(Sn_1, Sn_2, \dots)$ et si elle contient des liens potentiellement reliés au domaine thématique défini par l'ensemble des mots-clés.

Les poids ω_i ont étés fixés suivant un compromis entre les différents concepts auxquels sont associés chaque critère. Ces concepts sont d'ordre linguistique - quelle nuance attribuons nous entre le mot *souhaité* et *voulu* - et sont donc purement subjectifs. Ainsi, on peut raisonnablement considérer que les mots-clés S_i ont un poids deux à trois fois moins important que les K_i. De même il est naturel de considérer que $\omega_{Sn} = -\omega_S$. En effet, un poids négatif handicape le score total obtenu par une page si un mot Sn_i apparaît dans le texte. En agissant ainsi, si un mot S_i et un mot Sn_i apparaissent dans un texte, leurs effets respectifs s'annulent sur la fonction d'évaluation. Enfin, l'importance de l'évaluation des liens dans la fonction de fitness dépend du contexte de la recherche. Un poids important des liens dans l'évaluation aura tendance à éliminer des pages ayant peu de liens tandis que l'inverse rendrait plus difficile l'exploration en ne favorisant pas les documents susceptibles de mener à de bons résultats. On peut également signaler que ce poids permet de produire des résultats de type *pages hubs* (voir section 1.2.2) fournissant de nombreux liens ou non.

Tous ces éléments étant fortement subjectifs, nous avons arbitrairement fixé les valeurs suivantes reflétant les constatations décrites ci-dessus :

$$\begin{cases} \omega_K &= 3 \\ \omega_S &= 1 \\ \omega_{Sn} &= -1 \\ \omega_L &= 4 \end{cases}$$

Cette évaluation numérique permet de caractériser facilement l'évolution de la qualité des documents retournés en résultats. Cependant, elle comporte certains défauts qui nous ont conduit à élaborer une deuxième fonction d'évaluation. En effet, l'utilisateur ne peut agir ici que sur des listes de mots-clés. L'objectif de la méthode étant de préciser au maximum les souhaits de l'utilisateur, il est important de laisser plus de liberté dans la requête. Cette nouvelle requête doit posséder plusieurs critères de recherche dont l'importance doit pouvoir être librement définie par l'utilisateur en fonction de ses besoins.

3.3.2 Deuxième fonction d'évaluation : évaluation multicritère

La requête de l'utilisateur doit être transcrite dans le formalisme de l'optimisation afin de pouvoir appliquer des méta-heuristiques à notre problème. Comme mentionné au début de ce chapitre, par rapport aux requêtes classiques utilisées dans des moteurs de recherche à base d'index comme *Google* ou *Altavista*, le formalisme que nous développons

permet d'étendre très largement la définition de la requête de l'utilisateur. Les méta-heuristiques peuvent utiliser de nombreux critères qui ne sont pas du tout pris en compte par les moteurs classiques.

Nous définissons ainsi dans cette nouvelle fonction un grand ensemble de critères pour comparer les pages entre elles et établir la fonction d'évaluation. Nous reprenons les éléments de la requête de la fonction précédente en y rajoutant d'autres options. Ainsi, l'utilisateur doit premièrement fixer des mots-clés $(K_1, K_2, ...)$ comme cela se fait dans les moteurs classiques. C'est le seul critère qui doit être obligatoirement rempli puisque, comme cela a été mentionné dans la section précédente, ces mots-clés sont utilisés pour interroger les moteurs classiques. Tous les autres critères sont optionnels mais permettent de beaucoup mieux filtrer les réponses. La liste suivante décrit chaque critère disponible.

- C_{K1} : liste des mots-clés $(K_1, K_2, ...)$ spécifiant le document à rechercher. Ces mots seront utilisés pour interroger les moteurs de recherche classiques.
- C_{K2} : favorise les pages dans lesquelles la proportion de mots du texte correspondant aux mots-clés est importante.
- C_{K3} : spécifie que tous les $(K_1, K_2, ...)$ mots-clés doivent être présents.
- C_{K4} : favorise les pages dans lesquelles les mots-clés apparaissent dans le début du document.
- C_{K5} : favorise les pages dont les mots-clés sont représentés équitablement (mesure de l'entropie).
- $C_{K6...8}$: favoriser les pages comportant des mots-clés en gras (respectivement italiques, soulignés).
- $C_{K9...}$: favoriser les pages rapprochant des couples de mots-clés ; on peut spécifier par exemple que les mots-clés «algorithme» et «génétique» doivent apparaître proches l'un de l'autre dans le texte du document.
- C_M : indique que les mots-clés $(M_1, M_2, ...)$ doivent être retrouvés dans les documents résultats.
- C_{Mn} : indique que les mots-clés $(Mn_1, Mn_2, ...)$ ne doivent pas être présents dans les documents résultats.
- C_S : permet d'accroître l'importance des documents dans lesquels apparaissent les mots-clés $(S_1, S_2, ...)$.
- C_{Sn} : permet de minimiser l'importance des documents dans lesquels apparaissent les mots-clés $(Sn_1, Sn_2, ...)$.
- $C_{F1...}$: favoriser les pages comportant des fichiers images (respectivement : films, sons, PS, PDF, etc).
- C_{Size} : permet de favoriser les documents comportant beaucoup de textes

Afin de permettre à l'utilisateur de mieux préciser le contexte dans lequel il souhaite voir apparaître ses mots-clés (critères C_{K2} à C_{K9}), il est possible d'indiquer par exemple la proportion souhaitée de mots du document correspondant aux mots-clés, ou encore la proximité souhaitée entre plusieurs paires de mots. Dans les critères C_M, C_{Mn}, C_S et C_{Sn}, nous définissons plusieurs autres catégories de mots-clés indiquant la présence impérative ou prohibée de mots (respectivement $(M_1, M_2, ...)$ et $(Mn_1, Mn_2, ...)$), la présence souhaitée ou non de mots (respectivement $(S_1, S_2, ...)$ et $(Sn_1, Sn_2, ...)$). Le

contenu plus général du document peut être indiqué dans les critères $C_{F1...}$ à C_{Size} : il s'agit par exemple du type de fichiers référencés dans la page, ou encore de la taille du document.

L'utilisation de plusieurs critères dans la requête nous mène vers la résolution d'un problème multicritère similaire à ceux que l'on retrouve en optimisation. Il n'est donc plus possible de définir une valeur comme dans le cas de la fonction présentée dans la section précédente.

Chacune des options de notre requête utilisateur représente alors un critère d'évaluation pour la fonction d'évaluation. Nous cherchons ainsi à optimiser une fonction multicritère. Nous définissons alors la fonction d'évaluation utilisée dans notre modèle de manière à pouvoir établir une relation de dominance entre 2 points de notre espace de recherche. Le critère de dominance de *Pareto* est classiquement utilisé en optimisation pour résoudre ce problème. Il va par conséquent être utilisé pour établir la relation de supériorité entre 2 pages P_1 et P_2 de l'espace de recherche.

La notion de dominance au sens de Pareto se définit comme suit : on dit que le point P_1 domine le point P_2 si, $\forall i, f_i(P_1) \geq f_i(P_2)$ (avec au moins une inégalité stricte), où les f_i représentent les critères à maximiser. L'ensemble des points qui ne sont dominés par aucun autre point forme la surface de Pareto. Tout point P de la surface de Pareto est dit *optimal*, dans la mesure où on ne peut trouver un point de l'ensemble améliorant la valeur d'un critère de P sans diminuer la valeur d'au moins un autre critère.

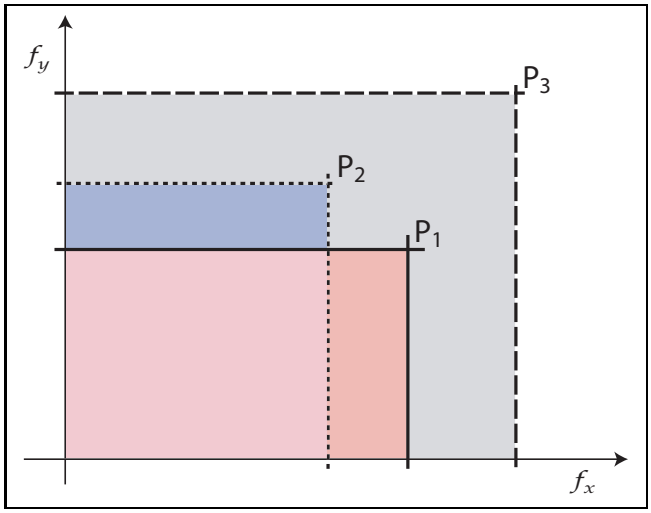


FIG. 3.4 – Exemple de dominance au sens de Pareto sur deux critères (x et y). Il est impossible de conclure sur la dominance du point P_1 sur le point P_2. Mais le point P_3 domine les deux autres points.

La figure 3.4 illustre un exemple de dominance au sens de Pareto sur deux critères, x et y. On constate que le point P_3 domine les deux autres points P_1 et P_2 sur les deux critères ($f_x(P_3) > f_x(P_1)$; $f_y(P_3) > f_y(P_1)$; $f_x(P_3) > f_x(P_2)$; $f_y(P_2) > f_y(P_2)$). P_3 est donc considéré comme optimal au sens de Pareto pour les critères x et y. Par

contre, il n'est pas possible de déterminer la dominance du point P_1 sur le point P_2 et réciproquement. En effet, chacun des deux points domine l'autre sur un unique critère (x pour P_1 et y sur P_2).

Comme on a pu le constater dans l'exemple ci-dessus, le critère de Pareto ne permet pas de conclure sur la dominance d'un point par rapport à l'autre. Pour résoudre ce problème, nous demandons à l'utilisateur d'indiquer l'importance relative qu'il donne à chaque critère que nous modélisons par des poids $(\omega_{C_K1}, \omega_{C_K2}, ...)$. La dominance de la page P_1 par rapport à la page P_2 sera alors déterminée par le calcul d'un score pour chacune d'elle. Ce score est calculé en effectuant la somme des poids $\omega_{C_.}$ pour tous les critères $C_.$ telle que P_1 domine P_2 pour $\omega_{C_.}$.

Par exemple, si la page P_1 domine P_2 sur le critère C_1 et P_2 domine P_1 sur le critère C_2. Pareto ne peut conclure. Les poids ω_{C_1} et ω_{C_2} ont été fixés au préalable à des valeurs respectives de 2 et 3. Par conséquent le score de P_1 est égal à $\omega_{C_1} = 2$ puisqu'il a dominé l'autre page sur le critère C_1 et le score de P_2 est égal à $\omega_{C_2} = 3$. Par conséquent, la page $P2$ est considérée comme meilleure que la page P_1. Ce dernier mode de calcul permet à coup sûr de déterminer la dominance d'une page sur l'autre ou l'égalité de chacune des pages.

Par conséquent, on ne dispose pas ici d'une évaluation numérique (i.e. de $S \rightarrow \mathbb{R}$) de chaque point de notre espace, mais d'une évaluation capable de déterminer si un point P_1 de S domine un autre point P_2 de S. Ceci est cependant suffisant pour pouvoir utiliser des algorithmes de recherche locale tels les algorithmes évolutionnaires ou des algorithmes à base de méthode tabou. On peut en effet toujours réduire la comparaison des points ou des individus traités par ces algorithmes à une notion de dominance à deux protagonistes.

Nous avons défini un modèle permettant de transformer le problème original sous forme d'un problème d'optimisation. L'utilisation de deux opérateurs de recherche O_{rand} et O_{explo} permet d'explorer cet espace de recherche de manière similaire à ce que l'on peut trouver en optimisation. Ces opérateurs sont utilisables dans tout type d'algorithme de recherche exploitant une méthode d'exploration locale. Ce modèle est de plus complété par deux fonctions d'évaluations adaptées au problème et tentant de récupérer un maximum d'information sur les besoins d'un utilisateur.

Cependant, l'utilisation de tels algorithmes de recherche apporte souvent un coût non négligeable en terme de puissance de calcul utilisée. Notamment, l'ensemble comprenant le téléchargement et l'évaluation d'une page représente un fort taux d'occupation du processeur. Il est par conséquent utile d'essayer de paralléliser la tâche voire de la distribuer sur plusieurs entités de calcul.

3.4 Présentation d'une architecture distribuée

La charge de calcul rencontrée dans les algorithmes de recherche d'information est très importante. Une solution à cette contrainte est sans aucun doute la parallélisation. Elle permet de répartir l'effort sur plusieurs machines et de limiter ainsi le temps nécessaire à l'évaluation globale de pages au cours de l'exécution d'une recherche.

On peut noter que les moteurs de recherche classiques utilisent les mêmes procédés afin de réduire le temps de construction de leur index et de répondre très rapidement aux milliers de requêtes simultanées dont ils sont la cible. Nous avons déjà abordé ce point dans la section 1.3 du chapitre 1. Le moteur de recherche *Google* utilisait notamment début 2001 plus de 15 000 machines connectées entre elles pour obtenir un temps de réponse moyen par requête inférieur à la seconde.

De telles ressources de calcul sont très coûteuses à entretenir. Il existe cependant une autre méthode permettant d'approcher voire de surpasser cette puissance. En effet, si l'on comptabilise les ressources non occupées des machines présentes sur Internet, on s'aperçoit aisément qu'un grand potentiel s'offre à nous. Des modèles de calculs distribués sur Internet utilisant les ressources laissées libres par les entités présentes sur le réseau ont ainsi vu le jour. On peut citer notamment des programmes analysant de grandes quantités de données comme *Seti@home* [Seti@Home] - pionnier dans ce domaine - s'occupant des bruits radios provenant de l'espace ou encore *Décrypthon* [Décrypthon] chargé de cartographier le génome humain.

Nous avons étendu notre modèle sur cette idée en élaborant en son sein une interface d'abstraction permettant de rendre indépendante l'*intelligence* de la recherche - autrement dit l'algorithme gérant la stratégie de recherche - des opérations plus basiques réalisant le téléchargement, l'analyse et l'évaluation des divers documents rencontrés. Il est ainsi très facile d'exporter ces dernières opérations sur des machines distantes et de faire communiquer l'ensemble des entités mises en œuvre au travers de l'interface ainsi construite. On peut ainsi facilement créer un architecture fortement distribuée sur Internet.

3.4.1 Interfaces d'abstraction mises en œuvre

Nous définissons ici les éléments utilisables par les algorithmes d'optimisation que nous allons utiliser afin de résoudre notre problématique. Le modèle que nous avons conçu est divisé en plusieurs modules remplissant chacun une fonction de base et interagissant entre eux au travers d'interfaces bien définies. Ces interfaces permettent en quelque sorte à chaque module d'exporter ses fonctionnalités et de proposer les siennes aux autres modules.

Nous avons ainsi défini plusieurs familles de modules présentées brièvement dans le tableau 3.2. Chacune a un rôle élémentaire à jouer dans le processus de recherche et utilise les fonctions offertes par les autres modules.

Cette architecture nous permet d'utiliser un mécanisme de parallélisation complexe sans pour autant à avoir à complexifier les algorithmes de recherche mis en œuvre.

Module	**Fonction**
Interface d'interrogation	Interface utilisateur (serveur web) permettant de formuler une requête au moteur de recherche
Algorithme de recherche	Algorithme gérant la stratégie de recherche utilisée
Moteur de recherche	Permet d'interroger les moteurs de recherche classiques afin de récupérer une liste de liens en résultat
Téléchargement	Le manager de téléchargement
Extraction	Analyse d'un document en fonction de son type (html, pdf, ...)
Évaluation	Évaluation d'un document
Cache	Cache permettant d'éviter de télécharger plusieurs fois la même page
Filtre	Sert à filtrer les pages à ne pas télécharger (pages interdites par les responsables de sites, limites de recherche, ...)
Résultat	Analyse des résultats fournis par l'algorithme de recherche (permet notamment d'établir une classification des pages)

TAB. 3.2 – Liste des principaux modules utilisés dans notre modèle.

Du point de vue de l'algorithme, seules les opérations de demande de téléchargement, analyse et évaluation de pages Web ainsi que les opérations de comparaisons des évaluations de deux documents sont accessibles. L'algorithme doit par conséquent se charger uniquement de gérer ces ressources mises à sa disposition.

3.4.2 Modèle distribué indépendant de l'algorithme de recherche

À partir des interfaces d'abstraction établies dans la section précédente, nous avons construit notre modèle en pensant à une architecture massivement distribuée. La partie distribuée est basée sur une architecture client/serveur qui permet de distribuer l'effort de téléchargement, analyse et évaluation des pages Web sur plusieurs clients distants. Cette architecture nous permet de tirer profit dans notre modèle d'algorithmes dans lesquels on peut identifier des opérations maître et des opérations esclaves. Par exemple, la parallélisation des AG présentée dans la section 2.1.3.1 utilise un tel mécanisme. La population est centralisée sur une ressource maître et plusieurs ressources esclaves la manipule indépendamment des autres.

Nous considérons ici que c clients (ordinateurs, processeurs ou threads d'exécution) ont été enregistrés sur un serveur et sont disponibles pour effectuer une recherche parallèle. Le modèle que nous avons adopté consiste à :

- centraliser l'algorithme principal sur le serveur,
- utiliser plusieurs threads d'exécution en parallèle sur ce même serveur effectuant des requêtes de téléchargement/évaluation,

– envoyer l'ensemble des opérations de téléchargement/évaluation à un *broker* qui répartit l'effort de recherche sur les c clients.

Le *broker* agit ici comme un *proxy* afin de faire le lien entre tous les éléments que composent notre architecture distribuée. Il se fait passer pour un serveur de recherche auprès des clients et pour un client aux possibilités très importantes auprès du serveur.

Un client reçoit les liens - ou URL - des pages à évaluer. Il télécharge alors les pages et les évalue. Une fois ce travail fait, l'évaluation est renvoyée au *broker* qui la retransmet au thread à l'origine de la requête. Ce mécanisme est traduit dans le schéma présenté dans la figure 3.5.

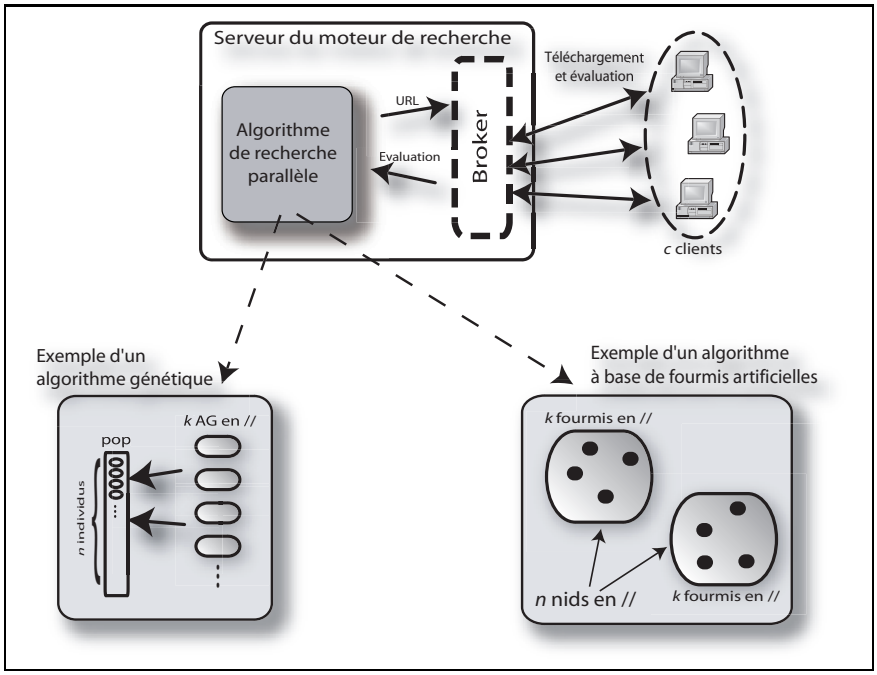


FIG. 3.5 – Architecture distribuée du modèle de recherche élaboré. Les demandes de téléchargement d'un algorithme quelconque sont envoyées à un *broker* qui les transmet à son tour à plusieurs clients indépendants qui téléchargent, analysent et évaluent les documents correspondants.

On peut également noter que chaque client est conçu de manière à pouvoir réaliser plusieurs opérations de téléchargement, analyse et évaluation en même temps. Ce point est relativement important parce qu'il permet d'exploiter efficacement le temps perdu pendant le téléchargement dû à la lenteur du réseau.

Ainsi, à partir du moment où un algorithme de recherche est capable d'effectuer plusieurs requêtes de téléchargement/évaluation en parallèle, cette architecture permet d'obtenir une application distribuée de manière transparente sans changer le fonctionnement de l'algorithme. Dans la mesure où un grand nombre de clients se connectent depuis Internet pour participer à l'effort de recherche, cette architecture devient massi-

vement parallèle. En quelque sorte, notre outil de recherche peut étendre facilement et dynamiquement les ressources matérielles dont il dispose.

3.5 Conclusion

Ce chapitre nous a permis d'établir le modèle générique d'un outil de recherche d'information sur Internet. De par la modélisation adoptée, nous avons transformé l'espace de recherche initial en un espace de recherche mathématique. Ceci nous permet de concevoir des algorithmes d'optimisation capables de retrouver et de présenter à un utilisateur des documents en fonction de ses souhaits exprimés dans une requête.

Nous avons alors défini deux opérateurs de recherche communément utilisés en optimisation : un opérateur de création heuristique qui permet de générer un point de notre espace de recherche et un opérateur d'exploration locale. Ces opérateurs permettent de parcourir l'espace à notre disposition de manière à retrouver les points intéressants du point de vue de l'utilisateur.

Deux fonctions d'évaluation de page Web ont été élaborées. Elles permettent de déterminer si un document est conforme ou non au souhait de l'utilisateur. Ces deux fonctions seront utilisées par la suite pour tenter d'évaluer les résultats obtenus par les algorithmes de recherche que nous avons conçus.

Ce modèle a également été conçu dans le but de distribuer l'effort de recherche sur plusieurs entités de calcul. Afin de faciliter l'élaboration de stratégies de recherche, nous avons fait en sorte de rendre cette parallélisation indépendante de l'algorithme utilisé dans le modèle. De cette manière, il est possible de paralléliser la recherche à la fois sur une machine en utilisant un parallélisme basé sur les threads - le transfert de documents depuis Internet s'effectuant en mode asynchrone - ou en utilisant des machines réparties sur un réseau local ou sur Internet, et ce de manière transparente pour l'algorithme de recherche.

Chapitre 4

Geniminer : un AG séquentiel pour la recherche d'information sur Internet

Résumé

Dans ce chapitre, nous présentons un algorithme génétique séquentiel pour la recherche d'information sur Internet : *Geniminer*. Cette méthode utilise le formalisme défini dans le chapitre précédent et cherche à optimiser une fonction d'évaluation appliquée sur des pages Web. Les individus sont considérés comme des documents contenant un texte et des liens. L'AG guide alors l'exploration locale en parcourant les liens hypertextes, et répartit l'effort de recherche sur les meilleures pages.

4.1 Motivations

Les algorithmes génétiques ont montré leur efficacité dans la résolution de nombreux problèmes et notamment dans les problèmes d'optimisation. La modélisation de l'espace de recherche que nous avons adoptée nous permet de tirer parti des propriétés des AG en optimisation dans notre problème.

Nous avons vu dans la section 2.4 du chapitre 2 qu'il existe plusieurs applications à base d'AG concernent le Web. On citera notamment [Sheth, 1994], [Fan *et al.*, 1999], [Morgan et Kilgour, 1996] et [Moukas, 1997] concernant la recherche d'information, [Vakali et Manolopoulos, 1999] dont l'AG gère l'accès à un cache pour accélérer l'accès à l'information ou encore [Monmarché *et al.*, 1999a] utilisant un AG interactif afin de générer un style de site Web.

Notre approche se distingue des autres dans la mesure où nous modélisons le problème de recherche d'information à un niveau proche d'un paysage de qualité (*fitness landscape*) : l'AG traite directement avec les points de l'espace de recherche, les pages

Web. Notre travail se distingue également des méthodes décrites dans la section 2.4.1 (*InfoSpiders*) par le fait que nous souhaitons faire intervenir l'utilisateur le moins possible afin de diminuer au maximum le temps requis pour l'analyse des résultats en automatisant complètement la recherche.

L'algorithme que nous avons mis au point combine les avantages d'une requête riche et filtrante avec la stratégie de recherche génétique. Il bénéficie du bon comportement des AG en ce qui concerne la résolution du dilemme d'exploration versus exploitation [Holland, 1975] : il décide quelles pages explorer en priorité et quelles pages éliminer (avant que tous leurs liens ne soient évalués).

4.2 Définition de l'algorithme

L'algorithme génétique que nous proposons ici s'inspire d'un modèle particulier d'algorithme génétique : l'AG *steady state*. Cette modélisation a été initialement introduite par Whitley [Whitley et Hanson, 1989] et a la particularité de ne générer qu'un unique individu à chaque itération de l'algorithme.

Nous allons par la suite présenter et définir l'ensemble des éléments composant cet algorithme : la modélisation des individus, la fonction de fitness utilisée et les opérateurs génétiques mis en œuvre pour explorer l'espace de recherche.

4.2.1 Modélisation d'un individu et de la fonction de fitness

Pour chaque problème, il est nécessaire d'adapter la représentation d'un individu au sein d'une population. Cette adaptation permet de faire converger l'AG plus ou moins rapidement et de tenir compte naturellement des contraintes de chaque modélisation d'un problème.

Dans le cas présent, nous utilisons le formalisme défini dans le chapitre précédent. L'espace de recherche que nous cherchons à optimiser est l'ensemble représenté par le graphe de liaison des documents d'Internet et par une fonction d'évaluation de chaque page Web. Internet est par conséquent un espace de recherche qui contient beaucoup de points (les pages Web), et qui est structuré comme un graphe avec des relations de voisinage (les hyperliens).

D'une manière globale, nous allons définir un individu de la population comme une page Web. Cet individu doit pouvoir être évalué numériquement par la fonction d'évaluation présentée dans la section 3.3.1. Cette fonction de fitness calcule la qualité d'une page Web en fonction de la requête de l'utilisateur. Elle prend en compte le texte, la typographie utilisée et les liens hypertextes contenus dans les documents.

L'individu ainsi généré peut être grossièrement représenté par un texte et par des liens vers les documents compris dans son voisinage direct.

4.2.2 Définition des opérateurs génétiques

Les opérateurs génétiques servent à diversifier la population gérée par l'AG afin d'explorer le plus efficacement possible l'espace de recherche. Nous avons vu dans le chapitre 2 que les AG disposent généralement de deux opérateurs : l'opérateur de croisement et l'opérateur de mutation. Cependant, dans le cas du Web, le graphe défini par les pages Web et leurs liens ne permet pas de définir facilement certains opérateurs de recherche. Il est en effet difficile de définir un opérateur de croisement ou de mutation comme on le conçoit classiquement dans les AG. En effet, en représentant un individu comme une page Web, il paraît inopportun de *mélanger* les termes d'un document avec ceux d'un autre. Le texte global perdrait alors tout son sens et l'évaluation en découlant ainsi que les individus résultats n'auraient que peu de valeur. De la même manière, un croisement des liens hypertextes tendrait à nuire à l'idée que le texte entourant un lien est un bon indicateur de la signification du document cible.

On peut toutefois noter qu'il serait possible de combiner les liens contenus dans deux pages parents P_1 et P_2 si ces deux pages ont des liens communs, ou des liens pointant vers le même serveur Web. On peut alors raisonnablement considérer qu'il est souhaitable de combiner ces informations en orientant la recherche vers ces liens ou ces serveurs Web communs.

Malgré tout, comme mentionné dans le chapitre précédent, au moins deux types d'opérateurs (création ou exploration locale) peuvent être définis pour explorer le Web avec un algorithme d'optimisation.

Nous utilisons ici un opérateur de création heuristique qui va donner une adresse de page Web en fonction des résultats donnés par les moteurs classiques (*Altavista*, *Google*, *Lycos*, *Teoma* et *Yahoo* dans nos expérimentations). La méthode utilisée consiste simplement à interroger chacun des ces moteurs avec les mots-clés $(K_1, K_2, ...)$ et à extraire les adresses données comme résultats. Ces interrogations de moteurs se font de manière incrémentale, au fur et à mesure que l'opérateur de création est appelé. Notons qu'il existe un très grand nombre de moteurs de recherche - une liste importante mais non exhaustive est donnée sur le site Web *www.finderseeker.com* - mais nous nous sommes concentrés pour le moment sur les plus généraux et les plus importants. L'opérateur de création permet donc de donner des points initiaux de bonne qualité dans la recherche génétique. Cet opérateur correspond à l'opérateur O_{rand} défini dans la section 3.2.1.

L'opérateur de mutation effectue tout d'abord une sélection par tournoi binaire au sein de la population. Pour cela, deux individus sont tirés aléatoirement et le meilleur des deux est choisi. À partir de cette page parent P, nous engendrons une page enfant E en explorant le voisinage local direct de P. Pour cela, les liens trouvés dans P sont triés par ordre décroissant suivant la valeur de la fonction $Link_{eval}$ décrite dans le chapitre précédent (voir section 3.3.1). De cette manière, les liens ayant le plus de chance de donner un enfant de bonne (ou de meilleure) qualité sont explorés en premier. Cet opérateur correspond à l'opérateur $O_{explo}(P, 1)$ dans le formalisme adopté au chapitre 3.

Il faut cependant noter que quel que soit l'opérateur utilisé, il se peut qu'aucun

nouvel individu ne puisse être produit. En effet, dans le cas de l'opérateur de création heuristique, les moteurs de recherche classiques fournissent des liens en un nombre limité (par exemple au maximum 1000 pour le moteur de recherche *Google*). Par conséquent, lorsque les résultats sont épuisés, l'opérateur O_{rand} est dans l'incapacité de fournir un nouvel individu. D'une manière similaire, l'opérateur O_{explo} génère un individu à partir du voisinage d'un point P particulier. Si tous les liens provenant de P ont déjà été explorés, ou si cette page ne possède pas de voisins, l'opérateur va également être dans l'incapacité de fournir un nouvel individu. Lorsque les deux opérateurs ne peuvent plus fournir de nouveaux individus, la recherche s'arrête.

4.2.3 Détails de l'algorithme utilisé

L'algorithme génétique utilisé ici est de type *steady state*. Cela signifie qu'à chaque itération, un unique individu est généré. Il s'agit par conséquent d'un AG séquentiel qui va guider l'exploration de l'espace de recherche en utilisant tour à tour les deux opérateurs mis à sa disposition. L'opérateur O_{rand} explore l'espace afin de trouver de nouvelles solutions tandis que l'opérateur O_{explo} exploite une zone locale de l'espace en tentant d'améliorer la qualité globale de la population.

L'algorithme 4.1 décrit le déroulement de *Geniminer*. Initialement, la population est vide et va croître petit à petit jusqu'à atteindre une taille maximale $Pop_{\max}$. Les premiers individus de la population sont générés par un opérateur de création heuristique d'individus O_{rand}. Cet opérateur interroge les moteurs classiques pour obtenir des pages Web intéressantes. À partir du moment ou suffisamment d'individus subsistent dans la population pour effectuer une sélection, l'algorithme va décider quel opérateur utiliser pour générer un nouvel individu grâce à une probabilité P_{explo}.

Les pages peuvent alors être sélectionnées suivant leur qualité f et peuvent donner des descendants soit par l'opérateur d'exploration locale O_{explo} (choisi selon une probabilité P_{explo}) soit par l'opérateur de création O_{rand}, selon une probabilité $(1 - P_{explo})$.

L'exploration consiste ici à choisir judicieusement un lien l sortant d'une page parent P et à proposer comme descendant la page indiquée par l (voir section 4.2.2). Les principes de l'algorithme génétique *steady state* font qu'à chaque génération un seul individu est engendré. Celui-ci est ensuite inséré dans la population s'il reste de la place ou s'il est meilleur que le plus mauvais individu de la population. En agissant de cette manière, les qualités moyenne, maximale et minimale de la population ne peuvent donc que croître dès que la population a atteint son nombre maximum d'individus.

Si plus aucun des moteurs ne fournit de réponse, alors l'opérateur de création n'est plus utilisé dans l'algorithme et on se sert à la place de l'opérateur d'exploration locale. Et de la même manière, lorsque l'on a exploré tout le voisinage d'une page P, on utilise l'opérateur de création à la place de l'exploration locale. Enfin, si aucun des deux opérateurs ne peut plus fournir de pages, la recherche s'arrête naturellement.

Intuitivement, l'algorithme de recherche va avoir un comportement allant du méta-moteur classique (avec $P_{explo} = 0$) à un moteur explorant le plus possible les liens

(1) Enregistrer la requête de l'utilisateur et Définir la fonction d'évaluation f
(2) $Pop \leftarrow \emptyset$
(3) **tantque** Condition de terminaison non rencontrée **faire**
(4) Engendrer un descendant E :
(5) **si** $(|Pop| < 2) \vee$ avec une probabilité $1 - P_{explo}$ **alors**
(6) $E \leftarrow O_{rand}$ (page issue des moteurs standards)
(7) **sinon**
(8) Choisir une page parent $P \in Pop$ par tournoi binaire et engendrer un descendant par mutation : $E \leftarrow O_{explo}(P, 1)$ (exploration des liens de P)
(9) **finsi**
(10) Evaluer $f(E)$
(11) **si** $(|Pop| < Pop_{\max}) \vee (f(E) > \min_{p \in Pop}\{f(p)\})$ **alors**
(12) Insérer E dans Pop
(13) **finsi**
(14) **fintantque**
(15) Pop est le résultat proposé à l'utilisateur.

Alg 4.1: Geniminer : un algorithme génétique *steady state* utilisé dans un moteur de recherche

trouvés dans les pages ($P_{explo} = 1$). En effet, en utilisant uniquement l'opérateur O_{rand}, seules les pages Web issues directement des moteurs de recherche sont analysées. C'est le principe d'un méta-moteur.

En fonction de la probabilité P_{explo}, la stratégie de sélection de l'algorithme génétique - utilisant les deux opérateurs O_{explo} et O_{rand} - va décider de la survie d'une page dans la population. Elle va ainsi également décider du fait que les liens trouvés dans cette page seront explorés avec plus ou moins d'intensité. En effet, plus un individu reste longtemps dans la population, plus il aura de chances d'être sélectionné par l'opérateur O_{explo} et donc de voir un de ses liens engendrer un nouvel individu.

Afin d'accélérer l'évaluation des pages et éviter de télécharger plusieurs fois les mêmes pages, nous ajoutons à ces deux opérateurs une *liste noire* de pages ayant déjà été explorées par l'algorithme. Ainsi, lorsque l'opérateur de création ou d'exploration locale engendre une page qui a déjà été visitée, celui-ci est relancé à nouveau pour obtenir une autre page. Cela permet de gagner du temps notamment dans les domaines où les liens sont souvent les mêmes.

De plus, notre outil peut utiliser un cache local stockant les pages récemment téléchargées. Cette technique permet d'accélérer significativement les performances de manière incrémentale lorsque deux requêtes thématiquement très proches se succèdent.

4.3 Évaluation de l'algorithme

Afin d'évaluer l'algorithme nous avons utilisé la première fonction d'évaluation décrite dans la section 3.3.1. Cette fonction produit pour chaque page analysée une valeur numérique dont l'intensité varie selon la correspondance de la page analysée à la requête formulée par l'utilisateur. Une valeur importante indique une bonne adéquation de la page à la requête.

Num	mots-clés $(K_1, K_2, ...)$	Mots souhaités $(S_1, S_2, ...)$	Mots non souhaités $(Sn_1, Sn_2, ...)$
1	telecom crisis analysis	mobile future	France
2	fiber optic technology information	network	
3	text poet flower wind	Baudelaire	Rimbaud
4	wine excellent price good	Bourgueil	Bordeaux
5	buy cd music michael jackson	download mp3	
6	mouse disney movie animation	DVD	
7	artificial ant algorithm		
8	genetic algorithm artificial ant	experimental comparison	
9	javascript window opener	tutorial free	
10	dll export class template	code example	

TAB. 4.1 – Requêtes de test utilisées dans l'évaluation de *Geniminer* pour la première fonction d'évaluation.

Nous avons implémenté l'algorithme décrit dans ce chapitre sur une machine de type PC (Celeron 500MHz, 128 Mo) et défini dix requêtes différentes (voir table 4.1) pour tester le comportement de notre moteur. Ces dix requêtes portent sur plusieurs domaines thématiques et représentent différents niveaux dans les besoins d'information d'utilisateurs potentiels. Elles sont certainement plus spécifiques que les requêtes le plus fréquemment utilisées dans les moteurs de recherche [Silverstein *et al.*, 1998] mais elles permettent de définir assez précisément un sujet.

4.3.1 Étude des paramètres

Dans l'algorithme que nous avons défini, il existe 3 principaux paramètres : la taille maximale de la population gérée par l'algorithme génétique, la probabilité d'exploration déterminant l'opérateur à utiliser à chaque itération, et le nombre de pages téléchargées et évaluées avant de considérer la recherche terminée.

Ce dernier paramètre est certainement le plus difficile à fixer. En effet, plus celui-ci est grand, plus la recherche va prendre du temps et laissera de chance à l'AG de trouver de meilleurs résultats. Cependant, afin de pouvoir établir une comparaison de toutes les approches que nous avons étudiées, nous avons décidé d'utiliser un protocole

de test et de comparaison similaire à celui utilisé dans la comparaison d'algorithmes d'optimisation. Pour cela, on donne à chaque version de l'algorithme le même nombre d'évaluations et par conséquent de téléchargements à effectuer.

Dans un premier temps, nous avons cherché à établir l'influence de la taille de la population. Pour cela, nous avons effectué des tests en modifiant la valeur de la taille de la population pour le téléchargement d'un nombre fixé de pages (1000 documents sont téléchargés). La table 4.2 montre l'évolution de la qualité des trente et des cinquante meilleurs individus de la population en fonction de la taille maximale de la population $Pop_{\max}$. On constate que si la population est trop importante, la sélection des individus par tournoi binaire ne concentre pas la recherche sur les pages les plus importantes. Mais d'un autre côté, si la population est trop petite, la recherche se focalise sur quelques individus et tend à décroître la qualité des résultats. Cette tendance a été confirmée sur plusieurs requêtes et il semble que l'utilisation d'une valeur $Pop_{\max} = 100$ soit un bon compromis.

Taille pop	**30**	**50**	**100**	**150**	**200**	**300**
30 premiers	398.0	411.5	457.9	429.5	427.1	428.5
50 premiers	n.a.	337.6	350.4	357.6	354.0	354.1

TAB. 4.2 – Moyenne des premiers éléments de la population après 1000 pages explorées suivant la taille de la population ($P_{explo} = 0.5$) pour la requête 2.

Le deuxième paramètre mis en jeu est la probabilité de mutation. Nous avons effectué, tout d'abord, une analyse du comportement de notre moteur pour une requête particulière pour analyser l'évolution de la recherche au cours du temps. Nous avons ainsi choisi aléatoirement la requête 3 (voir table 4.1) et avons effectué plusieurs tests en faisant varier le paramètre P_{explo}. Rappelons qu'avec $P_{explo} = 0$ notre moteur ne fait qu'utiliser les résultats fournis par les moteurs classiques. Il se comporte donc comme un méta-moteur. Plus P_{explo} augmente, plus l'algorithme met en oeuvre l'exploration du voisinage local des pages.

Sur la figure 4.1 nous montrons l'évolution de la qualité moyenne des individus. Le début de la courbe est bruité du fait que la population initiale ne contient que très peu d'individus. À partir de la génération 100, on constate que l'amélioration est progressive pour les différentes courbes présentées. On peut remarquer que plus la probabilité d'utiliser la mutation augmente, plus les résultats s'améliorent en moyenne. Bien sûr, si cette probabilité se rapproche trop de 1, la qualité des résultats chute. Nous retenons $P_{explo} = 0.5$ comme valeur assurant un compromis entre la recherche heuristique effectuée par les moteurs de recherche et la recherche locale guidée par les liens des individus.

On constate également deux zones distinctes de progression de la qualité de la population sur l'ensemble des valeurs de P_{explo} testées. En effet, avant environ le $600^{ième}$ téléchargement, les résultats issus des moteurs de recherche sont assez prolifiques et

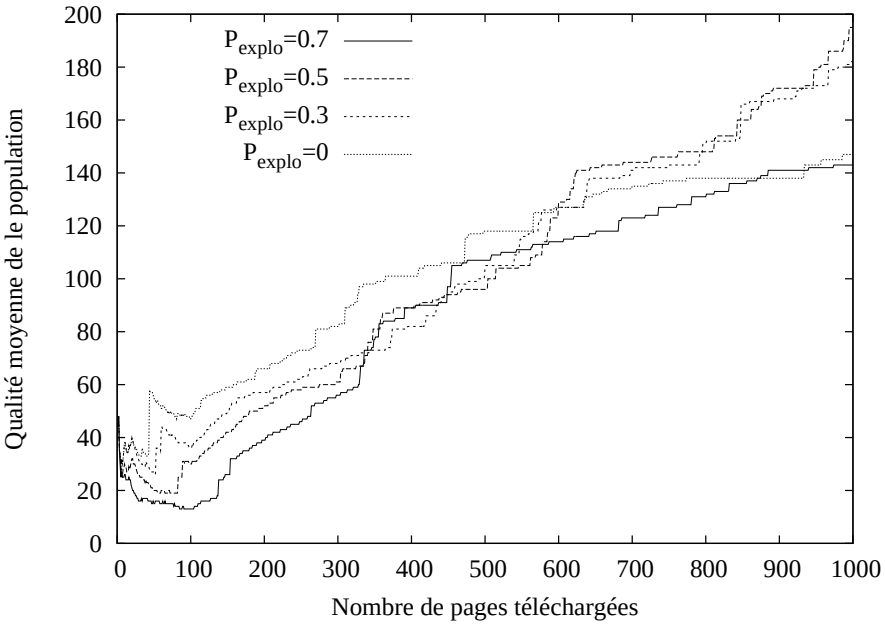


FIG. 4.1 – Évolution de la qualité moyenne de la population de *Geniminer* pour différentes valeurs de P_{explo} en fonction du nombre de pages téléchargées.

permettent d'obtenir une bonne qualité de la population. Après ce seuil, les résultats issus de ces moteurs tendent à s'épuiser et l'exploration locale montre alors son efficacité à trouver des individus de meilleure qualité.

4.3.2 Évaluation comparée

Dans la suite, nous avons testé l'AG sur l'ensemble des requêtes de test afin de juger de la qualité de *Geniminer*. Une fois de plus, nous avons utilisé le même protocole de test : chaque algorithme testé peut télécharger et évaluer 1000 pages Web. Cela représente environ entre une à quatre heures de temps de chargement et de calcul pour tester une requête. Dans toute la suite, la population a été limitée à $Pop_{Max} = 100$ individus à la suite des tests réalisés sur la taille de la population (voir table 4.2).

La table 4.3 montre les résultats obtenus par l'algorithme *Geniminer* sur les dix requêtes de tests utilisés en fonction de la probabilité d'exploration utilisée. Lorsque $P_{explo} = 0$, *Geniminer* se comporte comme un méta moteur de recherche analysant uniquement les résultats issus de moteurs de recherche classiques.

On constate ici que l'utilisation de l'algorithme génétique donne des résultats supérieurs pour 5 requêtes sur 10. Pour 4 autres requêtes, les meilleures pages trouvées par la méta recherche l'ont été aussi par *Geniminer*.

À partir de ces résultats, nous pouvons dire que les fonctions d'évaluation utilisées dans les moteurs classiques sont faites pour donner des réponses très rapides à des requêtes simples. Par exemple, elles ordonnent les pages sans tenir compte des mots-clés

R	GeniMiner ($P_{explo} = 0.5$)			GeniMiner ($P_{explo} = 0.3$)			Meta-recherche ($P_{explo} = 0$)		
	Min	Max	Moy	Min	Max	Moy	Min	Max	Moy
1	37.2	1034	83.2	39.5	1034	88.2	41.2	1034	**91.3**
2	102.7	810	212.9	109.8	825.5	**222.5**	99.3	794.5	205.4
3	48.8	1223	**195.7**	53.4	1839	182.1	54.6	1102	147.4
4	100.6	977.4	234.1	108.2	977.4	204.0	115.4	992.0	**243.3**
5	277.4	2229	**439.6**	219.8	2107	418.5	172.3	2107	375.2
6	150.3	841.4	256.7	151.8	1037	**284.3**	155.2	1037	281.7
7	63.7	575.3	**129.3**	59.7	575.3	120.3	51.4	575.3	96.6
8	122.7	1462	271.6	107.7	1510	273.5	129.3	1510	**310.3**
9	87.3	1097	198.2	78.8	1097	177.6	88.5	1097	**198.4**
10	99.2	2091	258.7	130.4	2091	328.8	141.5	3371	**430.3**

TAB. 4.3 – Évaluation de la qualité des résultats obtenus par l'utilisation d'une plus ou moins grande exploration locale dans *Geniminer*.

de la requête car l'analyse du contenu du document prendrait beaucoup trop de temps. Ces requêtes sont adaptées aux nombreux utilisateurs *grand public* d'Internet mais pas aux problèmes d'informations plus spécifiques et complexes comme cela est requis en veille stratégique. Si la fonction d'évaluation des pages utilisée ici est celle que l'utilisateur a en tête, alors il va passer beaucoup de temps à analyser les résultats des moteurs classiques avant de trouver l'information pertinente. Par contre, avec *Geniminer* l'information sera trouvée par l'utilisateur beaucoup plus rapidement. Mais pour réaliser cela, cet algorithme requiert un temps de calcul beaucoup plus important que les moteurs standards. C'est la raison pour laquelle nous considérons que notre approche est complémentaire des moteurs classiques car elle ne résout pas le même type de problèmes de recherche d'information.

On peut noter également dans ce tableau que les résultats donnés par l'utilisation des deux opérateurs, contrôlés par l'algorithme génétique, contiennent dans 9 cas sur 10 le meilleur élément. L'utilisation de l'algorithme génétique apporte un réel avantage en concentrant la recherche sur les points du graphe susceptibles de contenir de bonnes pages.

4.4 Conclusion

Le problème principal rencontré par cette approche est le temps nécessaire à la réalisation d'une recherche. En effet, il est nécessaire d'attendre entre une et quatre heures pour obtenir les résultats d'une requête. Cette attente est majoritairement due à la lenteur de téléchargement d'une minorité de pages Web. Les ressources de la machine effectuant la recherche sont alors très fortement sous-utilisés.

La fonction d'évaluation utilisée ici est également critiquable sur plusieurs points. Premièrement, cette fonction traduit la pertinence d'un document à une requête de recherche par une valeur numérique. Cette valeur est établie en sommant les fréquences des différents termes de la requête pondérés en fonction de leur type : les mots présents dans la liste K_i sont considérés de manière plus importante que les mots de la liste S_i et ainsi de suite. Par conséquent, si un document possède un grand nombre de mots-clés, celui-ci va obtenir un score très élevé. La moyenne de la population globale est ainsi fortement poussée vers le haut ce qui introduit un biais dans notre évaluation.

La requête utilisée dans cette évaluation reste relativement proche de celle employée dans les outils classiques de recherche. Seule la nuance entre plusieurs catégories de mots-clés est introduite et cette fonction optimise toujours la fréquence d'apparition des mots-clés dans les documents résultats. Or notre méthode prend de l'intérêt lorsque la définition de la requête d'interrogation du moteur est précise. Il est par conséquent plus approprié d'utiliser un système optimisant différents critères en fonction du souhait de l'utilisateur.

Malgré tout, les résultats obtenus dans nos expérimentations montrent que l'exploration de liens hypertextes améliore la qualité des pages selon la fonction d'optimisation prise en considération. Cependant, le système de recherche n'exploite pas les ressources matérielles mises à sa disposition en ne téléchargeant qu'un seul document à fois. Pour remédier à ce problème, nous présentons dans le chapitre suivant une parallélisation de *Geniminer* nommée naturellement *GeniminerII*. Cette méthode, bien que n'étant pas strictement équivalente à celle présentée dans ce chapitre, apporte des résultats similaires tout en diminuant considérablement le temps de recherche nécessaire à l'obtention des résultats.

Chapitre 5

GeniminerII : Parallélisation de la recherche génétique

Résumé

Dans ce chapitre, nous présentons *GeniminerII*, une évolution de l'algorithme *Geniminer*. Bien qu'il soit proche de ce dernier, il est construit sur la base d'un AG parallèle adapté à la recherche d'information. La parallélisation mise en œuvre dans le modèle permet de répartir la charge de téléchargement et d'analyse des pages Web sur plusieurs processus et/ou sur des machines distantes afin de réduire considérablement le temps de recherche de documents pertinents.

5.1 Une évolution naturelle de *Geniminer*

Comme nous l'avons signalé dans le chapitre précédent, le modèle d'AG utilisé par *Geniminer* utilise assez mal les ressources matérielles dont il dispose, que ce soit au niveau du processeur ou de la bande passante. Les protocoles et les algorithmes mis en jeu dans les réseaux n'allouent jamais - que ce soit du côté du client ou du serveur - la totalité de la bande passante à une unique opération afin de ne pas gêner l'ensemble des processus utilisateurs d'un réseau. C'est pourquoi beaucoup d'applications réseaux demandant une grande quantité de ressources réseau parallélisent leurs traitements afin de maximiser la bande passante mise à leur disposition.

Un moteur de recherche n'échappe pas à cette règle. Par ailleurs, de nombreux métamoteurs de recherche utilisent cette technique afin de garantir une meilleure disponibilité des résultats qu'ils peuvent fournir à leurs utilisateurs. Le but de notre méthode étant de parcourir un grand nombre de documents sur Internet le plus efficacement possible, nous avons de fortes demandes de bande passante. Il paraît ainsi naturel de paralléliser l'effort de recherche de *Geniminer* de manière à pouvoir proposer aux utilisateurs des

résultats dans le meilleur délai possible en utilisant le maximum de ressources à notre disposition.

GeniminerII a été principalement conçu dans ce sens. Cet algorithme a pour ambition de préserver les qualités de l'algorithme *Geniminer* tout en augmentant sa rapidité de fonctionnement et en maximisant l'utilisation des ressources matérielles mises à sa disposition.

La gestion principale de *Geniminer* est réalisée par un AG séquentiel. Il est possible de transformer cet algorithme en le parallélisant. Nous avons vu dans la section 2.1.3 que plusieurs modèles d'AG parallèles ont été proposés dans la littérature depuis le début du développement de ce type d'algorithmes. Dans notre problème, le point crucial est de paralléliser l'évaluation et le téléchargement des pages Web. À cause de la latence des réseaux, le temps de transfert des pages entre les serveurs Web et nos ordinateurs est très important en comparaison du temps requis pour comparer plusieurs individus et pour gérer la population génétique.

Le modèle algorithmique gérant une population d'individus centralisée est parfaitement adapté à ce cas. L'AG accède alors à la population en effectuant des opérations de lecture/écriture asynchrones.

L'utilisation d'une telle méthode parallélisée est très intéressante dans le cas où le temps nécessaire à l'évaluation d'un individu est très important comparé au temps requis pour l'exécution d'un opérateur. Comme nous avons pu le constater, notamment dans le chapitre précédent avec l'algorithme *Geniminer*, c'est tout naturellement le cas dans cette modélisation du problème.

5.2 Modélisation parallèle

GeniminerII utilise la modélisation présentée dans le chapitre 3. Dans ce modèle, il est possible d'utiliser une parallélisation distribuée sur plusieurs machines distantes. Pour cela, l'algorithme doit prendre en compte un certain formalisme de parallélisation. En effet, dans cette approche, nous avons séparé les opérations de manipulation d'une population d'individus des opérations de plus bas niveaux comme le téléchargement ou l'analyse de pages Web. Cela permet de concevoir des algorithmes de recherche indépendamment de la manière dont sont traitées les entités du problème à analyser.

Nous définissons ainsi dans la suite les éléments critiques formant un goulet d'étranglement dans l'algorithme initial *Geniminer* de manière à concentrer l'effort de parallélisation sur ces points.

5.2.1 Principes

Nous avons identifié dans la modélisation de résolution de la problématique un goulet d'étranglement : le téléchargement et l'analyse des pages Web. Ce sont par conséquent ces opérations qu'il est judicieux de paralléliser. Nous identifions ainsi chaque entité parallélisée en fonction des besoins d'évaluations d'individus dans notre algorithme. Ces opérations sont nécessaires à chaque génération d'individu et donc à chaque utilisation

de l'opérateur O_{rand} pour la création - et donc le téléchargement - d'un nouvel individu, et de l'opérateur O_{explo} qui explore un lien d'un individu de la population. Nous avons alors décidé d'utiliser une parallélisation à base de *threads* d'exécutions. Ce modèle se base sur la parallélisation des calculs dans les AG abordé dans la section 2.1.3.1. Chaque thread - ou entité parallélisée - doit être capable d'agir sur la population d'individus indépendamment des autres threads. Cette population est centralisée sur une ressource de calcul *maître* et l'algorithme principal de gestion de la population s'exécute sur cette machine. L'architecture utilisée permet alors tout naturellement d'exporter les opérations de téléchargement, analyse et évaluation sur d'autres ressources de calcul permettant de maximiser la bande passante à sa disposition.

Geniminer est contrôlé par un AG *steady state*. Dans ce type d'algorithme, un seul individu est généré à chaque itération de l'algorithme. Ainsi, pendant qu'un individu se crée grâce à l'action d'un des deux opérateurs O_{rand} ou O_{explo}, décrits dans la section 3.2 et choisis respectivement selon une probabilité $(1 - P_{explo})$ et P_{explo}, la population gérée reste inchangée. Dans le cadre d'un algorithme parallèle, où plusieurs individus sont générés en même temps, il n'est plus possible de garantir cette propriété. Nous avons donc fait évoluer l'algorithme de *GeniminerII*. Ce nouvel AG n'est pas équivalent à l'ancien mais reste néanmoins très proche de lui.

Nous avons également apporté certaines modifications par rapport à l'algorithme *Geniminer* afin de remédier à des défauts que nous avons pu constater. Ainsi, dans ce premier algorithme, la totalité des solutions est conservée dans la population de l'AG. Cependant, il est possible que l'algorithme ait exploré tous les liens d'une page pertinente au cours de son exécution. Il est nécessaire de conserver ce document dans les résultats mais ce dernier n'est plus d'aucune utilité au sein de la population. En effet, l'opérateur O_{explo} ne peut plus engendrer de nouvel individu à partir de lui, tous ces liens ayant déjà été explorés.

Pour améliorer l'AG, nous avons construit *GeniminerII* autour de deux populations de documents. Une population contrôlée par l'AG (Pop_G) qui évolue grâce à l'utilisation des opérateurs de recherche, et une population de résultats (Pop_R) qui ne contient que les pages les plus pertinentes à un moment donné de l'exécution de l'algorithme. Lorsqu'un document de la population de l'AG ne peut plus engendrer de descendance, tous ces liens ayant été explorés, celui-ci est éliminé de Pop_G mais conservé dans Pop_R. Inversement, lorsqu'un nouvel individu est généré, il est inséré dans les deux populations s'il est meilleur que le pire individu de Pop_G et de Pop_R respectivement. À chaque instant, les deux populations d'individus ne sont donc pas forcément équivalentes. Pop_G permet de guider la recherche tandis que Pop_R conserve les meilleurs individus rencontrés au cours de la recherche. On peut également noter que les tailles respectives de chaque population ne sont pas liées et qu'il est tout à fait raisonnable de considérer des dimensions très différentes l'une de l'autre.

L'accès aux ressources que représentent les deux populations d'individus Pop_G et Pop_R est contrôlé par l'emploi d'un sémaphore sem_{Pop} afin de ne pas corrompre les données enregistrées en leur sein. Cette protection garantit le bon état des opérations

d'entrée/sortie de sorte qu'à un instant donné, un seul thread d'exécution peut avoir accès ou peut manipuler les populations d'individus. Ce verrou n'handicape cependant pas l'exécution de l'algorithme car ces opérations sont de durée négligeable au regard du temps mis pour analyser une page Web.

Tous les mécanismes présentés ci-dessus sont résumés dans la figure 5.1. Elle illustre le fonctionnement global de l'AG parallèle utilisé dans *GeniminerII*. Dans ce schéma, seules les entités entourées en pointillés ne peuvent être manipulées de manière concurrente. Ainsi, les opérations d'insertion d'un individu dans une population et de sélection d'un individu dans Pop_G en vue de l'application de l'opérateur O_{explo} se déroulent de manière séquentielle, tandis que les opérations de création d'un individu par quelque moyen que ce soit, de choix d'un lien à explorer dans un individu, de comparaison et d'évaluation d'individus sont parallélisées.

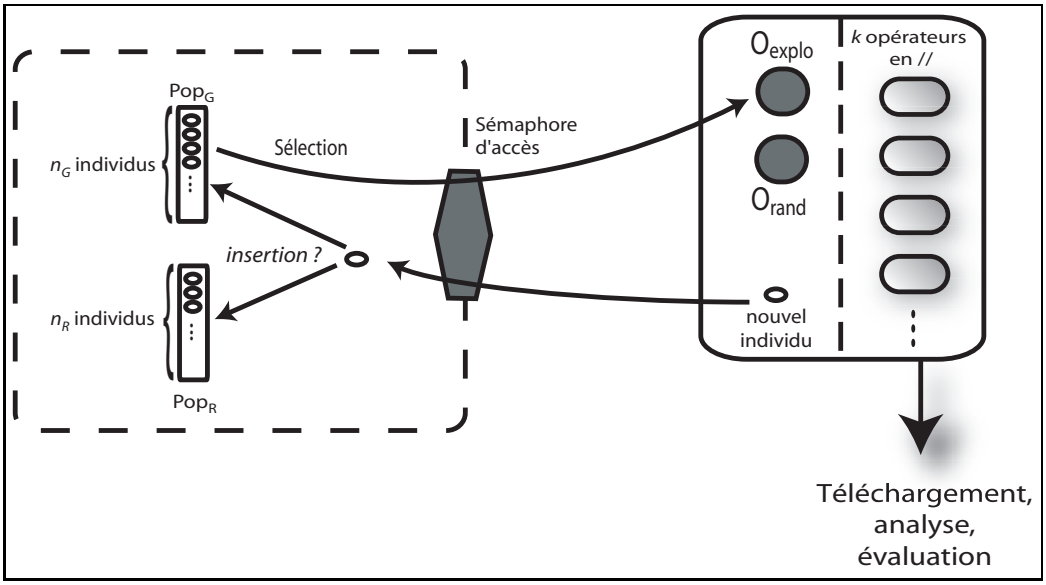


FIG. 5.1 – AG parallèle utilisé dans *GeniminerII*. Une population d'individus Pop_G est manipulée à l'aide de plusieurs opérateurs en parallèle tandis qu'une autre population d'individus Pop_R stocke les meilleurs éléments. L'accès aux deux populations est protégé par un mécanisme de sémaphores.

Enfin nous employons, de la même manière que dans *Geniminer*, une liste noire des pages à ne pas explorer. Cette liste contient les pages Web déjà visitées. Quatre sortes de pages sont par conséquent concernées : celles qui sont présentes dans la population, celles qui ont été éliminées par la présence de meilleurs individus dans la population, celles dont les liens ont tous été explorés ou celles dont le téléchargement pose un quelconque problème.

L'accès à cette liste noire est également protégé par un sémaphore d'accès car il s'agit d'une ressource partagée entre tous les threads d'exécution. Ce sémaphore est indépendant du sémaphore sem_{LN} protégeant les populations d'individus Pop_G et Pop_R

afin de garantir une meilleure parallélisation des tâches. Il est alors important de faire attention à ne pas engendrer un inter-blocage en donnant la possibilité à deux processus différents d'accéder alternativement à l'une puis l'autre sémaphore. En effet, s'il existe dans l'algorithme la possibilité de réserver les ressources sem_{Pop} puis sem_{LN} et également de réserver ces mêmes ressources dans un ordre inverse, les processus exécutant l'algorithme de manière concurrente peuvent se retrouver en situation d'inter-blocage. Afin d'éviter ce problème, nous n'autorisons pas la réservation de la ressource sem_{Pop} à l'intérieur d'un bloc protégé par sem_{LN}.

5.2.2 Algorithme

L'algorithme 5.1 décrit le fonctionnement interne de *GeniminerII*. Il reste très proche de celui de *Geniminer* décrit dans le chapitre 4.

La condition de terminaison est difficile à évaluer. Elle peut concerner un temps de recherche donné, l'utilisateur déterminant alors la durée pendant laquelle il estime obtenir une réponse satisfaisante, ou un nombre d'itérations fixe (noté $it_{\max}$) de l'algorithme. Il faut cependant signaler que le paramètre P_{explo} étant fixe dans cet algorithme, le nombre final de pages issues de moteurs de recherche par l'intermédiaire de l'opérateur O_{rand} est statistiquement égal à $P_{explo} \times it_{\max}$. Cette ressource n'étant pas inépuisable, il peut être judicieux d'utiliser un critère d'arrêt dépendant du nombre de téléchargements ou de requêtes formulées à un moteur de recherche. Signalons toutefois, que même si aucun moteur de recherche ne peut plus fournir de résultats à explorer, la recherche continue en utilisant uniquement l'opérateur O_{explo} qui va explorer les liens des pages Web présentes dans la population.

On peut cependant imaginer d'autres types de critères d'arrêt plus classiques en optimisation ou plus spécifiques à une optimisation par AG :

- si la meilleure solution n'a pas été améliorée depuis un certain nombre d'itérations,
- si peu d'individus de la population génétique sont remplacés en un temps ou un nombre d'itérations donné,
- lorsqu'il n'est plus possible de faire appel à l'opérateur O_{rand}.

Dans toutes les expérimentations que nous effectuerons par la suite, nous utilisons un critère d'arrêt dépendant du nombre total de téléchargements effectués. Ce critère d'arrêt a l'avantage de pouvoir être utilisé dans tout type d'algorithme de recherche employé pour résoudre ce problème. Il est ainsi plus aisé de comparer ces diverses méthodes entre elles.

5.3 Expérimentations

Tout comme *Geniminer*, *GeniminerII* possède deux paramètres : la taille de la population et la probabilité d'exploration. Dans un premier temps, nous avons cherché à analyser l'influence de chacun d'eux en fonction de la requête. Cependant, à la vue des constatations faites dans le chapitre précédent sur la fonction d'évaluation, nous avons décidé d'utiliser une fonction plus proche des souhaits d'un utilisateur. Celle-ci

(1) Enregistrer la requête de l'utilisateur et Définir la fonction d'évaluation f
(2) $Pop_G \leftarrow \emptyset$
(3) $Pop_R \leftarrow \emptyset$
(4) **tantque** Condition de terminaison non rencontrée **faire**
(5) **faire** en parallèle
(6) Engendrer un descendant E :
(7) **si** $(|Pop_G| < 2) \vee$ avec une probabilité $1 - P_{explo}$ **alors**
(8) $E \leftarrow O_{rand}$ (page issue des moteurs standards)
(9) **sinon**
(10) Choisir une page parent $P \in Pop_G$ par tournoi binaire et engendrer un descendant : $E \leftarrow O_{explo}(P, 1)$ (exploration des liens de P)
(11) **si** P ne possède plus de liens inexplorés **alors**
(12) Éliminer P de Pop_G
(13) **finsi**
(14) **finsi**
(15) Évaluer $f(E)$
(16) **si** $|Pop_R| < Pop_{R\max}$ **alors**
(17) Insérer E dans Pop_R
(18) **sinon**
(19) **si** $f(E) > \min_{p \in Pop_R}\{f(p)\}$ **alors**
(20) Éliminer $\min_{p \in Pop_R}\{f(p)\}$
(21) Insérer E dans Pop_R
(22) **finsi**
(23) **finsi**
(24) **si** E possède des liens inexplorés **alors**
(25) **si** $|Pop_G| < Pop_{G\max}$ **alors**
(26) Insérer E dans Pop_G
(27) **sinon**
(28) **si** $f(E) > \min_{p \in Pop_G}\{f(p)\}$ **alors**
(29) Éliminer $\min_{p \in Pop_G}\{f(p)\}$
(30) Insérer E dans Pop_G
(31) **finsi**
(32) **finsi**
(33) **finsi**
(34) **fin** parallélisation
(35) **fintantque**
(36) **retourner** Pop_R.

Alg 5.1: Algorithme principal de *GeniminerII*

comporte donc plus de possibilités et est paramétrable directement par l'utilisateur. Cette fonction a été présentée dans la section 3.3.2 du chapitre 3.

Le résultat d'une méthode de recherche d'information sur Internet est un ensemble de pages Web. L'évaluation de résultats d'une telle méthode consiste donc à analyser ces documents afin de déterminer la pertinence de la réponse à la requête initiale. Il n'est cependant pas possible de reproduire la même analyse qualitative que celle effectuée sur l'algorithme *Geniminer* dans le chapitre précédent. En effet, la fonction d'évaluation utilisée ici est une fonction multiobjectif. Il est, par conséquent, uniquement possible de définir une notion de dominance d'une page Web sur une autre, et non plus de quantifier cette dominance.

Dans les sections suivantes, nous allons ainsi exposer la méthode d'évaluation retenue afin de comparer les différents paramétrages de l'algorithme. Cette méthode sera reprise dans les chapitres suivants afin d'évaluer d'autres méta-heuristiques et de les comparer à *GeniminerII*. Par la suite, nous effectuerons une évaluation expérimentale des paramètres de l'algorithme dans le but de déterminer les meilleures valeurs possibles de ces paramètres de manière à obtenir un bon compromis quel que soit le paysage de l'espace de recherche optimisé.

Nous avons effectué les expérimentations sur cet algorithme avec dix requêtes utilisateur présentées dans la table 5.1. Les critères C_x sont ceux définis dans la section 3.3.2 du chapitre 103. Ces requêtes utilisent l'ensemble des critères définis et couvrent plusieurs domaines d'information distincts. Ces requêtes sont forcément loin d'être exhaustives mais nous permettent d'avoir une idée relativement précise du comportement des algorithmes de recherche que nous proposons.

5.3.1 Apport de la parallélisation au temps d'exécution

Le principal apport de la parallélisation réside dans le temps de calcul gagné. Nous avons cherché ici à étudier l'influence de la parallélisation dans le temps mis pour effectuer une recherche. La parallélisation que nous avons adoptée consiste à utiliser sur une machine plusieurs threads d'exécution effectuant le travail des opérateurs génétiques O_{explo} et O_{rand}. Nous avons testé l'influence de la parallélisation en faisant varier le nombre de threads utilisés pour la recherche, augmentant ainsi petit à petit le degré de parallélisation utilisé. Les résultats obtenus sont présentés dans le tableau 5.2. Dans ces tests, nous avons utilisé le critère d'arrêt défini plus haut : l'algorithme s'arrête une fois que 1000 téléchargements ont été effectués.

Ces résultats montrent bien l'influence de la parallélisation des téléchargements telle que nous l'avons décrite dans les sections précédentes. Le temps de recherche diminue de manière régulière au fur et à mesure que nous augmentons le nombre de threads s'exécutant de manière concurrentielle. C'est un résultat qui paraît cohérent si l'on prend en considération les protocoles de gestion de réseau couramment utilisés dans les systèmes d'exploitation modernes (par exemple le protocole *TCP* défini dans la RFC 793). De plus lors de certains téléchargements, des serveurs Web peuvent fournir des pages avec de très faibles débits. Cela handicape naturellement le temps total mis par l'algo-

req	C_K	C_S	C_{Sn}	C_M	**autres critères**
1	buy cd music michael jackson	download mp3			C_{K1} C_{K3} $C_{K9}(4,5)$ C_{F3}
2	ant algorithm genetic	termit			C_{K1} C_{K3} C_{K4}
3	technology fiber optic information	network	crisis		C_{K3} C_{K4} C_{F6} C_{F8} $C_{K9}(2,3)$
4	javascript window opener	tutorial free		code	C_{K2} C_{K3} C_{K7} C_{Size}
5	mouse disney movie animation	DVD			C_{K1} C_{K3} C_{K4} C_{F2}
6	text poet flower wind	Baudelaire	Rimbaud		C_{K2} C_{K4} C_{K6} C_{K8} C_{F3}
7	tv reality show	internationnal			C_{K3} C_{F1} C_{F2} C_{K5} $C_{K9}(2,3)$
8	dll export class template	code example			C_{K1} C_{K3} $C_{K9}(1,2\|3,4)$
9	wine excellent price good	Bourgueil	Bordeaux		C_{K2} C_{K4} C_{K6}
10	genetic algorithm artificial ant	experimental comparison			C_{K4} C_{K5} $C_{K9}(1,2\|4,2\|3,4)$

TAB. 5.1 – Les 10 requêtes utilisées dans nos tests.

# thread	2	4	6	8	10	12	14
temps (sec)	3449	1704	1454	1234	1229	1143	1055

TAB. 5.2 – Temps d'exécution en fonction du nombre de threads d'exécution pour 1000 téléchargements (tests réalisés sur la requête 8).

rithme car les ressources du moteur de recherche sont en attente de ces pages et sont, par conséquent, beaucoup moins efficaces. Mais lorsque plusieurs threads de téléchargements s'exécutent en même temps, ce handicap s'amenuise car il reste statistiquement en permanence des ressources libres pour traiter de nouvelles pages Web.

Outre le temps d'exécution, il est nécessaire de valider l'algorithme en examinant son paramétrage et son efficacité dans la résolution du problème posé. Les sections suivantes sont dévolues à expérimenter cet aspect de l'algorithme *GeniminerII*.

5.3.2 Modèle d'évaluation

Comme indiqué dans la section 5.2.2, nous utilisons dans toutes les expérimentations effectuées un critère d'arrêt dépendant du nombre total de téléchargements réalisés par l'algorithme. Ce critère a l'avantage d'être indépendant de la méta-heuristique choisie

et nous permet donc de comparer aisément plusieurs méthodes entre elles. Elle permet également de se comparer équitablement avec un méta moteur de recherche, n'examinant et ne classant que les résultats fournis par des moteurs de recherche classiques.

Nous avons fixé ce seuil de téléchargement à 1000 pages Web pour toutes les exécutions d'algorithmes effectués avec cette fonction d'évaluation multicritères. Ce nombre de téléchargements permet à l'algorithme de recherche d'explorer une zone non négligeable d'Internet sans comparaison possible avec la gigantesque proportion de téléchargements effectués par les index des moteurs de recherche. Il est à noter que la même opération effectuée sans outil automatique par un utilisateur est beaucoup trop fastidieuse et prendrait beaucoup de temps.

Nous avons précédemment signalé que la méthode d'évaluation multicritère utilisée ne permet pas de quantifier la dominance d'un document sur un autre mais simplement de déterminer quelle page domine l'autre. Pour comparer les résultats - correspondant à un ensemble de pages Web - de plusieurs méthodes entre elles, nous considérons les n_r premières pages trouvées par chaque méthode pour une requête donnée et une exécution donnée. Nous fusionnons ensemble les pages trouvées par les m méthodes testées (soit $m \times n_r$ pages) et les trions selon la fonction d'évaluation. Un tri correspond simplement à une succession de comparaisons binaires d'éléments, le critère de dominance utilisé suffit donc à réaliser l'opération. Nous affectons alors à chaque page Web un score sc_p qui est inversement proportionnel à son rang dans le classement établi. Par exemple, sur un total de 30 documents, la page de rang 1 recevra un score $sc_{p_1} = 30$ alors que la page de rang 20 recevra un score $sc_{p_{20}} = 10$. Nous totalisons ensuite les scores obtenus par chaque méthode, c'est-à-dire que nous sommons les scores des pages obtenus en résultat d'une méthode particulière afin de déterminer le score pour cette méthode. Nous normalisons enfin les sommes obtenues afin que la meilleure méthode obtienne un score de 100.

Cette méthode nous permet d'avoir une comparaison objective entre plusieurs outils de recherche. Elle présente l'avantage de ne pas être sensible à la différence de grandeur des critères utilisés pour évaluer les pages. De plus elle privilégie - pour une même requête - une méthode fournissant un ensemble de bons résultats à une autre ne possédant que quelques très bons résultats confondus dans de mauvaises pages. Cela se justifie en considérant qu'un utilisateur recherche une méthode lui garantissant le meilleur ensemble global de résultats.

5.3.3 Étude des paramètres

Dans cette section, nous étudions l'influence mutuelle des deux paramètres de l'algorithme. Nous avons sélectionné à cette fin quatre requêtes pour illustrer les différents comportements que nous avons pu observer. Il s'agit des requêtes 1, 2, 4 et 7 présentées dans le tableau 5.1.

Pour chacune de ces requêtes, nous avons effectué des tests en faisant varier à la fois la taille de la population génétique (Pop_G) et la probabilité d'exploration P_{explo} gérant l'utilisation des opérateurs O_{explo} et O_{rand}. Rappelons ici que lorsque P_{explo} prend une

valeur de 0, cela signifie que seul l'opérateur O_{rand} est utilisé et que par conséquent seuls les résultats fournis par les moteurs de recherche sont analysés, à la manière d'un méta-moteur.

Nous avons échantillonné chacun de ces paramètres sur sept valeurs créant ainsi 49 configurations possibles. La taille de population génétique varie de 25 à 175 individus par tranche de 25 et P_{explo} varie, quant à elle, de 0 à 0.6 par pas de 0.1. Pour chacune de ces configurations, nous avons déroulé l'algorithme sur 1000 téléchargements. À partir des résultats obtenus, nous avons voulu examiner la population de résultats Pop_R sous trois points de vus différents. Ainsi, nous avons tout d'abord examiné - et mis en concurrence - les 30 premiers résultats obtenus sur une exécution, puis les 60 premiers résultats et enfin les 100 premiers résultats. Cela nous permet d'avoir une vision de la répartition de la qualité des résultats en portant notre regard des meilleurs individus à l'ensemble des résultats fournis par l'algorithme de recherche.

Nous avons également effectué trois exécutions pour chaque point de configuration. Nous appliquons alors notre algorithme de comparaison sur les 49×3 exécutions de l'algorithme et faisons une moyenne pour chaque point des trois résultats obtenus.

La figure 5.2 montre les résultats obtenus sur la première requête testée. La figure *a)* présente les moyennes obtenues sur les 30 premiers individus de la population de résultats, la figure *b)* ceux obtenus sur les 60 premiers résultats et enfin la figure *c)* concerne les 100 premiers résultats.

Tout d'abord, d'une manière globale, les trois courbes présentent relativement les mêmes tendances. À savoir une progression de la qualité des résultats avec une croissance de la valeur du paramètre P_{explo} quelle que soit la taille de la population choisie. D'une manière générale, sur toutes les courbes prises en compte, les plus grandes valeurs de P_{explo} et les plus petites et plus grandes valeurs de taille de population sont avantagées pour cette requête.

Pour des valeurs de P_{explo} allant de 0 à 0.3, on peut également considérer que les résultats sont globalement similaires sur les trois courbes présentées. On observe ensuite une progression avec une augmentation de la valeur de P_{explo} notamment lorsque l'on considère les scores obtenus en prenant en compte les 60 premiers résultats. Mais si on observe la courbe *c)* prenant en compte les 100 premiers résultats pour chaque configuration testée, on remarque un certain seuil maximum atteint pour une valeur de $P_{explo} = 0.5$. En effet, on observe alors que pour de petites et de grandes tailles de populations, on obtient les meilleurs résultats sur cette requête.

Bien que les résultats soient globalement uniformes suivant les valeurs de la taille de la population, on note tout de même quelques variations. On observe que sur cette requête, les petites, et dans une moindre mesure les grandes populations sont avantagées. Les plus petits scores sont obtenus par des tailles de populations de 125 individus. Cette remarque se vérifie d'autant plus lorsque l'on considère des moyennes prenant en compte un grand nombre d'individus, comme c'est le cas sur la figure *c)*.

Le sujet traité par cette requête concerne beaucoup de pages accessibles depuis Internet. Il est donc relativement facile pour un indexeur de fournir un grand nombre de réponses à la requête. Il en résulte que le tri nécessaire pour retrouver parmi ces réponses

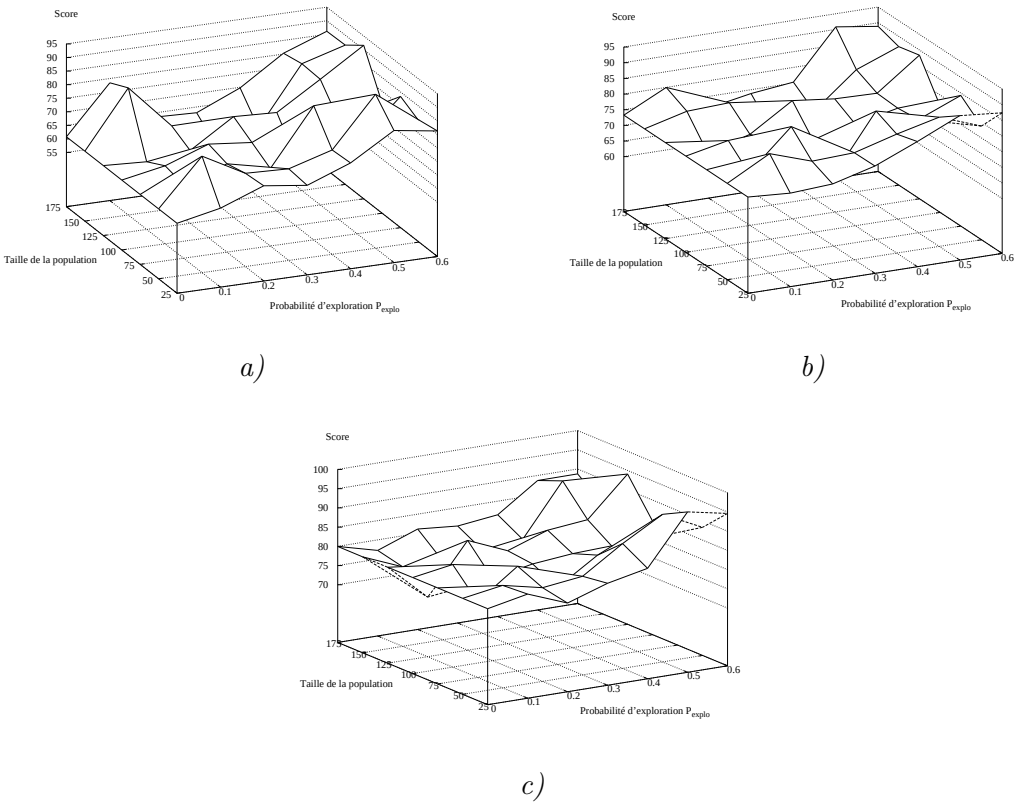


FIG. 5.2 – Influence conjointe de la taille de Pop_G et de P_{explo} pour l'algorithme *GeniminerII* pour la requête 1 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

les documents correspondant au souhait de l'utilisateur peut s'avérer fastidieux s'il est réalisé manuellement. Cependant, on constate que l'utilisation de l'opérateur d'exploration O_{explo} contrôlé par l'algorithme génétique augmente sensiblement la qualité des réponses retournées. Cela ne signifie pas que les moteurs de recherche traditionnels ne contiennent pas les pages retournées par *GeniminerII* mais plutôt que les pages considérées les plus pertinentes par notre fonction d'évaluation - construite en fonction du sujet exprimé précisément par l'utilisateur - ne sont pas classées en tête des réponses fournies. *GeniminerII*, en se laissant guider par l'AG arrive cependant à retrouver de telles pages Web en suivant les liens hypertextes dans le voisinage desquels figurent des éléments formulés dans la requête de l'utilisateur.

La figure 5.3 montre également le même type de résultats que précédemment mais avec des tests effectués sur la requête 3.

Ces courbes montrent de manière flagrante - surtout au regard des figures *a)* et *b)* - une progression constante de la qualité des résultats en fonction de valeurs grandissantes du paramètre P_{explo}. La courbe *c)* présente la même tendance mais de manière beaucoup

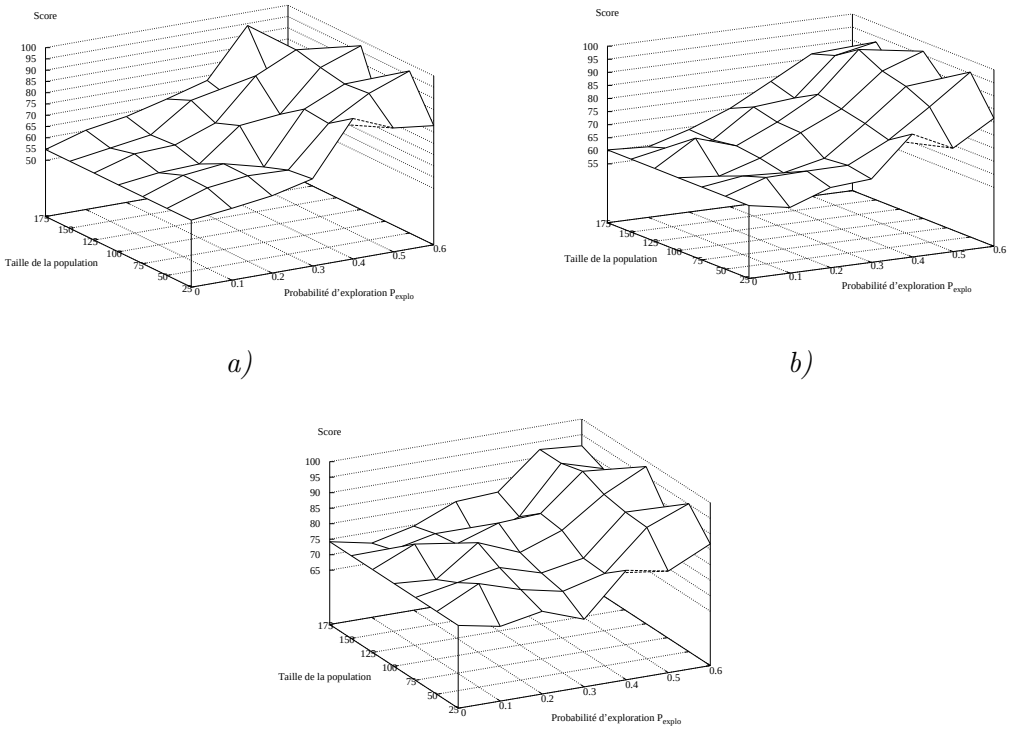


FIG. 5.3 – Influence conjointe de la taille de Pop_G et de P_{explo} pour l'algorithme *GeniminerII* pour la requête 3 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

plus atténuée, surtout pour de faibles valeurs de P_{explo}. Cela signifie que pour des valeurs importantes de P_{explo}, les tout premiers résultats sont bien meilleurs que ceux proposés par les moteurs de recherche classiques. Cependant, si l'on regarde un plus grand ensemble de résultats (les 100 premiers) pour chaque méthode, les bons résultats commencent à apparaître.

Ici encore on remarque, mais d'une manière beaucoup plus marquée que pour la requête précédemment étudiée, que le paramètre gérant la taille de la population génétique n'apporte que peu d'influence sur la qualité des résultats excepté pour de fortes valeurs de P_{explo} lorsque l'on regarde les premiers individus fournis en résultats de la recherche. Dans ce cas, une trop grosse population nuit à la recherche en ne concentrant pas assez les meilleures pages Web trouvées afin de garantir une exploration efficace des liens hypertextes à disposition.

Cette requête traite d'un sujet très technique et très précis. Les moteurs de recherche classiques ne fournissent pas énormément de résultats à ce type de requête. On peut raisonnablement considérer, par conséquent, qu'il n'existe pas beaucoup de documents traitant ce sujet sur Internet, du moins relativement au nombre total de pages Web

présentes sur ce réseau. Cependant, puisqu'il s'agit d'un sujet précis, plus la requête de l'utilisateur est riche, plus facile est la recherche de l'information. Les moteurs de recherche n'utilisent pas un langage leur permettant d'intégrer autant d'information. C'est pour cette raison que *GeniminerII*, en utilisant avec insistance l'opérateur O_{explo}, accède plus rapidement aux pages les plus pertinentes au sens de la fonction d'évaluation construite.

La figure 5.4 montre l'évolution de la qualité des résultats de la quatrième requête en fonction de la variation des paramètres testés.

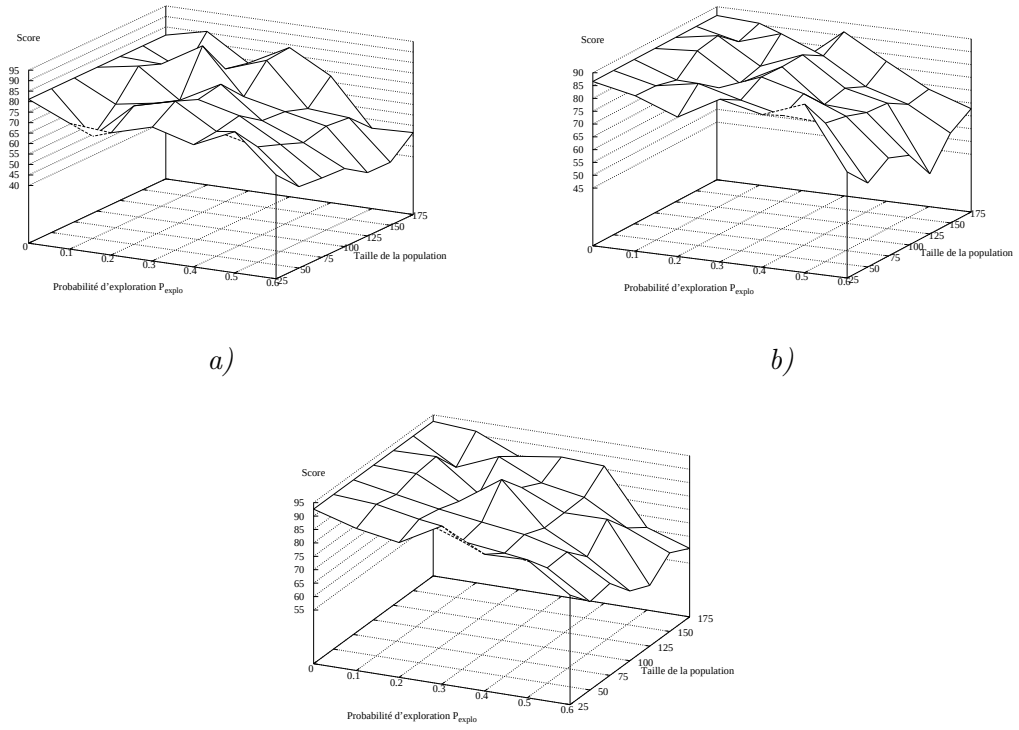


FIG. 5.4 – Influence conjointe de la taille de Pop_G et de P_{explo} pour l'algorithme *GeniminerII* pour la requête 4 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

Comme on peut le voir sur les courbes, la tendance est ici à l'inverse de ce que nous avons observé dans les deux requêtes précédentes. En effet, l'utilisation de l'opérateur O_{rand} uniquement, autrement dit en ne regardant que les pages Web fournies par les moteurs de recherche classiques, sans explorer de liens hypertextes, amène de meilleurs résultats que si on utilise l'opérateur O_{explo}.

On constate cependant que tant que P_{explo} ne prend pas de valeur supérieure à 0.3, les résultats obtenus sont relativement similaires entre eux. Mais au-delà de cette limite,

les résultats s'écroulent de plus en plus vite à mesure que P_{explo} augmente.

Cette requête apporte peu de résultats dans les moteurs de recherche et est assez mal traitée sur Internet. En effet, il existe extrêmement peu de pages Web apportant une réelle réponse complète au problème posé. La plupart des *"moins mauvais"* documents retournés le sont donc par les moteurs de recherche classiques et l'exploration de liens n'a plus alors beaucoup d'intérêt dans un tel cas. C'est principalement cette raison qui explique la faiblesse des résultats obtenus par des valeurs importantes de P_{explo} sur cette requête.

Encore une fois la faible influence du paramètre P_{explo} indique que la taille de la population n'affecte pas beaucoup les résultats. On peut cependant noter que de faibles populations génétiques favorisent de meilleurs résultats pour cette requête.

Le dernier cas que nous allons maintenant examiner se rapproche du cas précédent. Il s'agit de l'étude de la requête 7 présentée dans la figure 5.5.

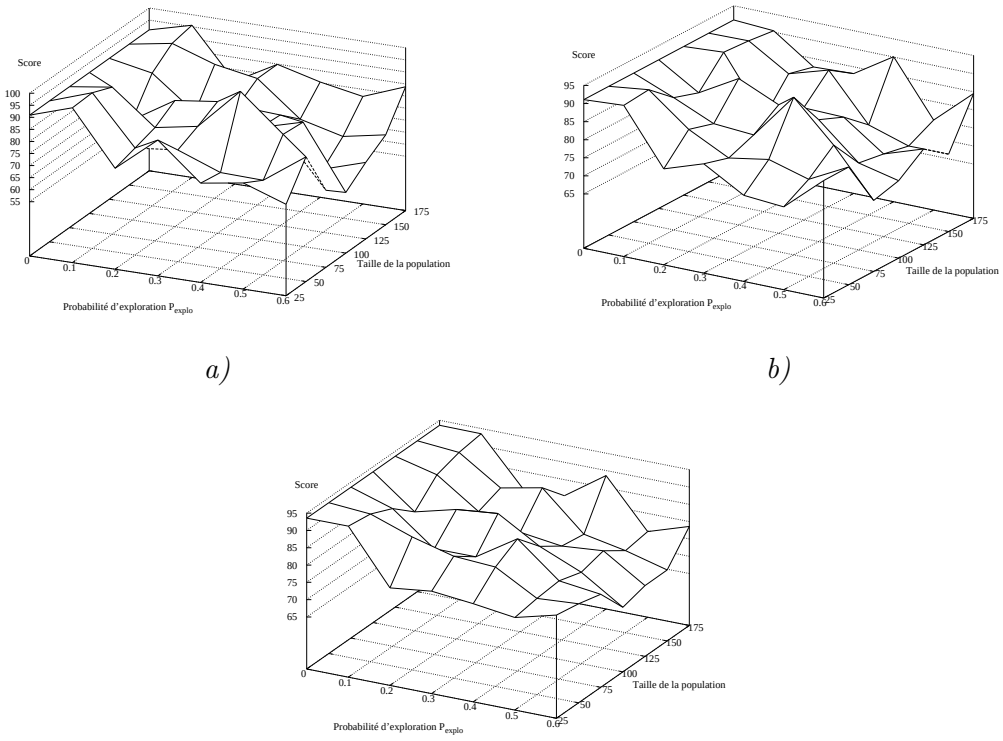


FIG. 5.5 – Influence conjointe de la taille de Pop_G et de P_{explo} pour l'algorithme *GeniminerII* pour la requête 7 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

Sur cette requête, on constate une tendance à augmenter la pertinence des résultats avec une probabilité d'exploration qui diminue. Cette diminution de pertinence est

constante avec toutefois une légère stabilisation pour la dernière valeur de P_{explo} testée : 0.6. Cette remarque est surtout valable sur les résultats des figures *b)* et *c)* prenant en compte un plus grand nombre de résultats que la figure *a)*.

Dans les trois précédentes requêtes analysées la valeur du paramètre $Pop_{G\,\max}$, gérant la taille de la population génétique, n'influençait que très peu les résultats. Pour la première fois, on peut remarquer dans les tout premiers résultats fournis par *GeniminerII* sur la figure 5.5 *a)* qu'une trop faible ou trop grande population nuit à la qualité des réponses. La valeur $Pop_{G\,\max} = 100$ forme une légère crête et on peut distinguer sur cette crête un pic pour une valeur $P_{explo} = 0.4$. Ce pic se retrouve également aux alentours de ce point, notamment pour des valeurs $P_{explo} = 0.3$ et $Pop_{G\,\max} = 25$ mais dans une moindre mesure. Il est difficile de trouver une explication à ce pic. D'autant qu'il diminue fortement lorsque l'on regarde un plus grand ensemble de résultats jusqu'à pratiquement s'atténuer dans la figure *c)* portant sur une analyse des 100 premiers résultats.

Cette requête fournit beaucoup de résultats dans un moteur de recherche classique. Il s'agit d'un sujet général dont les résultats pertinents sont présents en abondance. Par conséquent un index, ayant une vision globale d'Internet, a plus de chance d'obtenir de bons résultats qu'une méthode d'exploration qui peut *se perdre*, noyée dans une multitude de liens paraissant potentiellement intéressants.

D'une manière générale, sur cet ensemble de requêtes représentatives des observations réalisées, on peut dire que la taille de la population génétique Pop_G influence peu les résultats obtenus en comparaison avec l'action du paramètre P_{explo}. Ce dernier paramètre s'exprime différemment suivant le paysage de l'espace de recherche rencontré. Lorsque peu de pages Web traitent du sujet recherché ou ne le traitent que vaguement, il existe beaucoup moins de liens pointant vers ces *bons* documents - au sens du souhait de l'utilisateur - dans des pages sur le réseau. Par conséquent, une méthode de recherche à base d'index est très appropriée pour trouver cette information *rare*. Cependant, dans le cas contraire, notre évaluation très précise caractérise beaucoup mieux les pages pertinentes. Ainsi une recherche locale suivant efficacement les liens rencontrés dans les pages du Web arrive à de très bons résultats.

La taille de la population influence d'avantage les résultats lorsque le paramètre P_{explo} prend de grandes valeurs. Cela paraît logique parce que seul l'opérateur O_{explo} se sert de la population pour générer un nouvel individu. Cette influence est toutefois très limitée, comme nous l'avons déjà signalé sur les exemples détaillés.

Nous cherchons à obtenir des valeurs de paramètres représentant le moins d'inconvénient possible quel que soit le type de paysage rencontré. Par conséquent nous avons décidé de fixer dans la suite des évaluations une valeur $Pop_{G\,\max} = 100$ ainsi qu'une valeur du paramètre d'exploration de $P_{explo} = 0.6$.

5.3.4 Comparaison avec *Geniminer*

Afin de vérifier si les changements introduits par la parallélisation dans l'algorithme de *GeniminerII* ne nuisent pas à la qualité des résultats, nous avons comparé les deux

Requête		**Méta-**	**GeniMiner**		**GeniMinerII**	
n^o	**# res**	**recherche**	$P_{mut} = 0.5$	$P_{mut} = 0.3$	$P_{mut} = 0.5$	$P_{mut} = 0.3$
1	30	91.37	90.82	64.19	▶ 100	92.84
	60	92.61	97.22	80.73	98.81	▶ 100
	100	98.23	92.02	93.20	95.05	▶ 100
2	30	87.69	84.42	80.68	70.85	▶ 100
	60	94.79	85.86	▶ 100	68.20	92.71
	100	99.61	91.37	95.08	80.46	▶ 100
3	30	78.31	97.47	81.72	▶ 100	99.52
	60	90.77	94.56	89.84	99.36	▶ 100
	100	92.51	94.04	96.57	▶ 100	98.63
4	30	90.66	▶ 100	98.86	83.48	84.81
	60	90.42	95.90	▶ 100	91.18	83.12
	100	94.84	▶ 100	98.36	87.44	87.47
5	30	92.27	▶ 100	98.23	96.93	94.91
	60	94.70	94.34	96.20	99.11	▶ 100
	100	86.82	90.24	95.18	98.49	▶ 100
6	30	83.24	81.92	77.41	79.75	▶ 100
	60	▶ 100	86.76	98.56	87.79	99.34
	100	▶ 100	78.98	91.38	82.05	94.73
7	30	63.82	67.22	74.45	▶ 100	95.03
	60	77.92	77.72	85.52	▶ 100	95.78
	100	80.85	82.00	89.30	82.00	▶ 100
8	30	97.14	▶ 100	87.92	96.41	87.96
	60	82.37	86.23	81.82	▶ 100	88.99
	100	88.77	97.18	86.75	99.30	▶ 100
9	30	87.93	▶ 100	88.69	84.14	96.18
	60	▶ 100	96.51	91.86	88.05	87.27
	100	▶ 100	95.20	98.18	78.86	89.95
10	30	▶ 100	81.75	96.95	80.09	82.21
	60	▶ 100	80.01	95.68	78.78	92.67
	100	▶ 100	85.48	97.66	88.06	96.30

TAB. 5.3 – Comparaison entre *Geniminer*, *GeniminerII* et un méta-moteur sur 10 requêtes.

méthodes entre elles en utilisant les dix requêtes listées dans le tableau 5.1 de la page 138. Nous utilisons ici l'évaluation multicritères car elle prend mieux en compte dans la requête les souhaits formulés de l'utilisateur. Ces évaluations sont établies en comparaison avec une méthode étalon : l'utilisation unique de l'opérateur O_{rand}. On obtient ainsi une simple compilation des résultats fournis par les moteurs de recherche classiques à la manière d'un méta-moteur. Ce méta-moteur est parallélisé de la même manière que l'a été *GeniminerII*.

Nous avons résumé les résultats obtenus dans le tableau 5.3. Pour chaque requête, nous avons comparé les résultats obtenus par les trois méthodes avec différents paramètres. *GeniMiner* et *GeniMinerII* sont testés avec une exploration plus ou moins importante des liens locaux ($P_{mut} = 0.3$ et $P_{mut} = 0.5$).

La comparaison entre ces méthodes est réalisée sur les n_r=30, 60, 100 premiers résultats obtenus. Ceci nous permet de mieux quantifier la distribution des bons résultats : une méthode peut obtenir les meilleurs résultats sur les 10 premières pages, mais des résultats faibles en moyenne.

Nous pouvons immédiatement remarquer dans le tableau 5.3 que *GeniMiner* et *GeniMinerII* dominent les résultats obtenus par la méta-recherche sur 7 requêtes de nos tests, et que nos heuristiques obtiennent les 30 meilleurs premiers résultats pour 9 requêtes sur 10. Il apparaît que l'utilisation de ces méthodes pour la recherche d'information semble être plus profitable que la simple compilation de résultats obtenus par des moteurs de recherche standards à base d'index.

Le temps mis par le méta-moteur de recherche parallèle et *GeniMinerII* sont similaires pour chaque requête. Cela signifie que notre outil réduit certainement beaucoup le temps requis pour trouver une information pertinente. L'exploration locale des liens est un des avantages majeurs des méta-heuristiques, qui traitent naturellement le dilemme de l'exploration vs exploitation (tester des liens inconnus ou utiliser des liens produits par des moteurs de recherche standards).

La parallélisation utilisée dans *GeniMinerII* augmente légèrement la qualité des résultats obtenus par *GeniMiner*. En effet, pour 7 requêtes, l'algorithme parallèle obtient de meilleurs résultats globaux que l'algorithme séquentiel.

5.4 Conclusion

L'algorithme *GeniminerII* a rempli tous les objectifs que nous lui avions fixés. Il a amélioré grandement le temps d'exécution de *Geniminer* sans diminuer la qualité des résultats obtenus. Ces résultats ne sont toutefois pas surprenant car les deux algorithmes ont une très grande similarité de fonctionnement.

Cet algorithme offre également deux possibilités de parallélisation grâce à son implémentation au sein du modèle que nous avons conçu, présenté dans le chapitre 3. En effet, d'une parallélisation sur une machine basée sur des threads d'exécution, on passe aisément à une parallélisation sur des machines distantes sans intervenir sur l'algorithme de recherche lui-même.

Chapitre 6

Modélisation de plusieurs méta-heuristiques : *Antsearch* et *Tabusearch*

Résumé

Dans ce chapitre, nous exposons deux nouvelles méta-heuristiques de recherche d'information sur Internet. La première, *Antsearch*, s'inspire de l'algorithme *API*, une méta-heuristique basée sur le comportement de fourragement de fourmis réelles. Cette méthode permet de valider l'utilisation de notre modèle sur des algorithmes de types différents. Cependant, elle possède un très grand nombre de paramètres qui sont relativement difficiles à fixer. La deuxième méta-heuristique se base sur un algorithme tabou et, au contraire de la première ne possède qu'un nombre limité de paramètres. Les expérimentations réalisées sur les deux algorithmes nous ont permis de déterminer les meilleures configurations possibles pour l'ensemble des requêtes testées.

6.1 Vers une modélisation de plusieurs heuristiques

La modélisation adoptée pour résoudre le problème de recherche d'information sur Internet autorise l'emploi de tout algorithme d'optimisation utilisant un espace de recherche et des opérateurs. Les AG ont montré leur efficacité dans ce domaine, mais d'autres méta-heuristiques obtiennent de très bons résultats.

Dans le chapitre 2, nous avons notamment décrit une méta-heuristique d'optimisation basée sur le comportement de fourragement observé chez les fourmis réelles : *API*. Cette méthode a la particularité d'explorer un espace de recherche à la fois du point de vue global et local. Nous avons par conséquent décidé de créer un algorithme d'exploration d'Internet en s'inspirant d'*API* que nous avons appelé *Antsearch*.

Dans le chapitre précédent, nous avons vu que la parallélisation de la méthode de recherche permet d'améliorer grandement le temps dévolu à la recherche. Les algorithmes de fourmis sont construits autour de petits algorithmes manipulant des entités simples. Il est, par conséquent, très facile d'identifier des parties de l'algorithme de recherche pouvant être parallélisées. Nous avons ainsi immédiatement parallélisé *Antsearch* de manière à diminuer fortement le temps passé à la recherche de documents pertinents. Nous décrirons précisément cette méthode dans la section suivante.

Nous avons également voulu examiner le comportement sur notre problématique d'algorithmes très classiques en optimisation. Nous avons ainsi créé une nouvelle méta-heuristique (*Tabusearch*) basée sur une recherche tabou. Nous avons décrit la méthode initiale dans la section 2.2. Elle est relativement simple à formuler dans notre problème car elle ne nécessite qu'une notion de dominance d'un point sur un autre contrairement à d'autres méthodes où il est nécessaire de pouvoir quantifier cette notion de dominance.

6.2 Une approche à base de fourmis : *Antsearch*

Toutes ces implémentations de méta-heuristiques bien différentes les unes des autres montrent que le modèle que nous avons conçu (voir chapitre 3) est suffisamment générique pour permettre l'implémentation d'un grand nombre de méta-heuristiques d'optimisation pour résoudre le problème de recherche d'information sur Internet.

Les algorithmes à base de population de fourmis artificielles sont apparus relativement récemment au regard des algorithmes d'optimisation existants. Ce type d'algorithme s'est cependant développé rapidement et a par ailleurs trouvé de nombreuses applications dans bien d'autres domaines. Nous avons déjà abordé ce sujet dans la section 2.3.2 du chapitre 2.

L'optimisation est l'un des domaines dans lesquels ces algorithmes ont pu montrer tout leur potentiel de résolution. C'est donc tout naturellement que nous avons décidé de construire l'algorithme *Antsearch* puisque notre modèle s'appuie sur une transformation en problème d'optimisation. L'espace de recherche que nous générons est relativement complexe et l'adaptation naturelle des algorithmes basés sur le comportement de fourmis réelles convient parfaitement à ce type de paysage de recherche.

De plus, comme nous l'avons signalé dans le paragraphe précédent, ces méthodes ont également l'avantage d'être relativement faciles à paralléliser du fait de leur conception autour de petites entités indépendantes les unes des autres.

6.2.1 Une adaptation de l'algorithme API

L'algorithme *Antsearch* que nous proposons ici est une adaptation de l'algorithme API présenté dans la section 2.3.3.2 au contexte de la recherche d'information sur Internet.

API s'inspire du comportement de fourragement des fourmis de la famille *Pachycondyla apicalis*. Ces fourmis vivent au sein de colonies peuplées de quelques dizaines

d'individus seulement et possèdent une mémoire leur permettant de se souvenir de sites de chasse dans lesquels la récolte de nourriture a été fructueuse. Elles explorent leur environnement à la recherche permanente de nourriture pour nourrir la colonie. Leur stratégie de recherche est relativement simple mais efficace puisqu'elle leur a permis de traverser les âges et d'être encore présentes à l'époque actuelle.

D'un point de vue global, les fourmis explorent aléatoirement un point central dans l'espace de recherche représenté par le nid de la colonie de fourmis. Elles recherchent des points de bonne qualité en mémorisant des sites de chasse situés autour du nid dans une amplitude maximale A_{site}. Périodiquement, le nid de la colonie est déplacé vers un nouveau site de chasse et toutes les fourmis de la colonie oublient leurs connaissances acquises des *bons* sites de chasse afin de ne pas se perdre dans le nouveau paysage qui s'offre à elles.

D'un point de vue local, chaque fourmi se déplace autour du nid et mémorise p sites de chasse indépendamment des autres fourmis. Les alentours de chaque site de chasse sont explorés dans une amplitude maximale A_{local} afin de trouver de la nourriture. Dans notre modélisation, nous considérons que la capture d'une proie est représentée par une amélioration de la fonction à optimiser. Ainsi, la fourmi cherche en permanence à trouver de meilleurs points dans l'espace de recherche dans lequel elle évolue. Afin d'explorer plus efficacement une zone de l'espace de recherche fructueuse, lorsqu'une fourmi trouve une proie, elle va systématiquement chercher à explorer de nouveau le même site de chasse à la prochaine sortie de son nid. Dans le cas contraire, lorsqu'aucune proie n'a pu être détectée dans une zone de l'espace de recherche, la fourmi choisit d'explorer un des p sites de chasse qu'elle a mémorisé précédemment. De plus, afin de ne pas pénaliser la recherche sur des zones infructueuses, la fourmi a la possibilité d'oublier un site de chasse si celui-ci ne mène pas à la capture de nouvelles proies. Dans ce cas, le site oublié est remplacé par un nouveau choisi aléatoirement autour du nid de la colonie dans une amplitude maximale A_{site}.

Le comportement détaillé de chaque fourmi a déjà été exposé dans la section 2.3.3.2 du chapitre 2. Comme on vient de le voir, chaque fourmi possède une mémoire pour stocker les sites de chasse intéressants pour sa recherche de proie. Cette mémoire a une taille limitée par un paramètre p. Les fourmis possèdent également une patience d'exploration de site P_{local}. Lorsqu'une fourmi i, à partir d'un site de chasse donné, effectue $e_i > P_{local}$ explorations successives infructueuses, le site de chasse est considéré infructueux et est abandonné au profit d'un nouveau site.

De manière à pouvoir simuler API dans notre environnement, nous considérons une population F de $N_{fourmis}$ fourmis qui sont réparties de manière égale dans N_{nids} nids. Les nids et les sites de chasse sont représentés par des pages Web de notre espace de recherche Internet. L'amplitude maximale des mouvements (A_{site} et A_{local}) correspond dans notre modèle à l'amplitude maximale d'exploration locale depuis un point de notre espace et donc à un nombre de liens maximum à suivre pour générer un voisinage d'une page donnée. Autrement dit, cela correspond au deuxième paramètre de l'opérateur

O_{explo} que nous avons décrit dans la section 3.2.2.

Lorsqu'une fourmi f_i explore un site de chasse s_k, elle génère une page s'_k en utilisant l'opérateur $O_{explo}(s_k, A_{local})$. Si s'_k domine s_k, ce point devient un site de chasse en remplaçant dans la mémoire le site s_k : $s_k \leftarrow s'_k$, la localisation du site de chasse a changé. Enfin, lorsque f_i a effectué plus de $P_{local}(f_i)$ explorations infructueuses d'un site de chasse s_k, ce site est oublié et un nouveau site de chasse est généré en utilisant l'opérateur d'exploration locale afin de déterminer un site dans le voisinage du nid N. On utilise ainsi $O_{explo}(N, A_{site})$.

Pour pouvoir explorer au mieux l'espace de recherche, le nid est périodiquement déplacé. Nous avons défini deux conditions pour effectuer cette opération : lorsque les fourmis d'un nid ont effectué plus de $MaxDepFou$ mouvements d'une part et lorsqu'il n'existe plus aucun lien à explorer dans les sites de chasse aux alentours du nid d'autre part. Lors d'un déplacement de nid, nous choisissons selon une probabilité P_{creat} si la génération du nouveau site est opérée par l'opérateur O_{rand} qui interroge les moteurs de recherche classiques (décrit dans la section 3.2.1), ou selon une probabilité $(1 - P_{creat})$ si le nid est déplacé sur la meilleure page détectée dans les alentours du nid depuis son dernier déplacement. Dans tous les cas, la mémoire des fourmis du nid est réinitialisée avec de nouveaux sites de chasse localisés à proximité du nouveau nid.

L'algorithme API a également été étendu de manière à pouvoir simuler plusieurs nids en parallèle, de manière indépendante les uns des autres, et également de pouvoir simuler en parallèle plusieurs fourmis au sein d'un nid. Nous reviendrons plus en détail sur cette parallélisation dans la description algorithmique d'API dans la section suivante.

6.2.2 Algorithmes

Cette section décrit algorithmiquement la méta-heuristique de recherche d'information *Antsearch* basée sur le comportement de fourragement observé chez les fourmis réelles. La plupart des algorithmes à base de fourmis réelles sont parallèles d'un point de vue mathématique. Mais ils sont généralement simulés d'une manière séquentielle tant du point de vue logiciel que matériel. Nous avons parallélisé *Antsearch* selon une architecture réellement parallèle.

Au niveau global, l'algorithme 6.1 décrit la base de la simulation de la colonie de fourmis. Les fourmis recherchent des proies autour de leurs nids de manière indépendante les unes des autres. Dans cet algorithme, l'initialisation des fourmis correspond à la mise à zéro de la mémoire ainsi que du compteur d'échec. Ce compteur $(e_{i,j})$ sert à comptabiliser le nombre d'explorations successives infructueuses du site de chasse s_j par la fourmi f_i.

La structure logique de l'algorithme laisse ressortir deux entités qu'il semble logique de paralléliser : les nids des fourmis et les fourmis elles-mêmes. Par conséquent, nous avons décidé que chaque nid et sa population de fourmis doivent être simulés en parallèle des autres, sans aucune interaction entre elles. Plusieurs fourmis cohabitent sur chaque nid, il est donc également intéressant de paralléliser cet ensemble de manière à avoir

(1) Générer N nids
(2) Initialiser chaque fourmi
(3) **faire** en parallèle pour chaque nid (N_n)
(4) Choisir aléatoirement l'emplacement initial du nid : $N_n \leftarrow O_{rand}$
(5) **tantque** La condition d'arrêt n'est pas vérifiée **faire**
(6) **faire** en parallèle $\forall i / f_i \in N_n$
(7) FOURRAGEMENT(f_i,N_n)
(8) **fin** parallélisation
(9) **si** le nid doit être déplacé **alors**
(10) Selon une probabilité P_{creat}, $N_n \leftarrow O_{rand}$
(11) Selon une probabilité $(1-P_{creat})$, $N_n \leftarrow s^+$ /* meilleure solution atteinte */
(12) Vider la mémoire de toutes les fourmis : $\forall i \in F$, $m_{f_i} \leftarrow \emptyset$
(13) **finsi**
(14) **fintantque**
(15) **fin** parallélisation
(16) **retourner** les meilleures pages visitées durant la recherche.

ALG 6.1: Algorithme principal de *Antsearch*

une exécution réellement concurrente des fourmis de la colonie toute entière.

Cependant, si chaque entité est parallélisée, le système peut s'écrouler très rapidement. En effet, le nombre de threads mis en jeu devient vite très important en augmentant le nombre de nids et de fourmis utilisés. Cette limite de parallélisation dépend de l'architecture matérielle mise en jeu. Si une seule machine est utilisée dans le modèle que nous avons proposé dans le chapitre 3, les ressources disponibles seront nécessairement moins importantes que si l'on utilise une architecture distribuée sur plusieurs machines. Le nombre d'entités parallélisées sera par conséquent limité. D'une manière similaire, le nombre de téléchargements simultanés opérés par l'algorithme dépend de la quantité de bande passante disponible pour une machine et de la charge supportée par le processeur durant un téléchargement. Ce paramètre est a priori indépendant du nombre de threads utilisés pour l'exécution de l'algorithme. En effet, les fourmis peuvent effectuer des opérations ne nécessitant pas de téléchargements et utiliser, par conséquent, des ressources non exploitées par le téléchargement et l'évaluation de pages Web.

Pour ces raisons, nous avons décidé d'utiliser une architecture parallèle à deux niveaux. Un premier niveau, que l'on peut qualifier de haut niveau sert à exécuter l'algorithme contrôlant le fourragement des fourmis de manière concurrente (voir l'algorithme FOURRAGEMENT 6.2) tandis qu'un deuxième niveau - qualifié de bas niveau - permet de gérer l'ensemble des processus de téléchargement, d'analyse et d'évaluation des documents, de manière concurrente également. Nous paramétrons ainsi le nombre de fourmis agissant de manière concurrente à tout instant donné et le nombre de téléchargements, d'analyses et d'évaluations simultanés possibles.

Chaque nid est donc parallélisé et possède plusieurs fourmis. Mais le nombre d'exécutions simultanées de l'algorithme gérant les fourmis est limité et peut par conséquent ne pas correspondre au nombre de fourmis présentes. Dans ce cas, les fourmis sont considérées par les threads d'exécutions d'un nid comme des ressources à exploiter. Les fourmis sont par conséquent classées dans une liste en FIFO et dès qu'un thread a fini de traiter une exécution de l'algorithme FOURRAGEMENT pour une fourmi, il la place dans la liste et se charge d'exécuter le fourragement de la prochaine fourmi.

Les téléchargements sont quant à eux gérés de manière indépendante de chaque nid. L'algorithme principal dispose d'un *pool* d'entités de téléchargement, analyse et évaluation qui sont considérés comme des ressources partagées par l'ensemble des nids et par conséquent par l'ensemble des threads d'exécution de chaque nid. Cet ensemble est protégé par un sémaphore autorisant à un instant donné uniquement l'exécution d'un certain nombre de téléchargements correspondant au paramètre fixé dans l'algorithme. Ces ressources sont placées dans une liste gérée en FIFO et sont attribuées au fur et à mesure des demandes provenant des nids. Lorsqu'une demande est assouvie, la ressource est placée en queue de liste prête à être servie à un nouveau thread d'exécution provenant d'un nid quelconque.

Ce mécanisme de parallélisation à deux niveaux est illustré par la figure 6.1.

Le comportement local de chaque fourmi est donné par l'algorithme 6.2. Cette procédure, appelée FOURRAGEMENT, utilise le formalise suivant :

- m_{f_i} correspond au nombre de sites de chasse mémorisés par la fourmi f_i,
- p correspond au nombre maximal de sites de chasse mémorisable par une fourmi,
- s_j correspond au site de chasse j (au document j),
- S_{f_i} correspond à l'ensemble des sites de chasse mémorisés par la fourmi f_i,
- $e_{i,j}$ est un compteur d'échecs successifs d'exploration autour d'un site de chasse s_j par la fourmi f_i,
- $f(s_j)$ est la fonction d'évaluation appliquée au document j représenté par le site s_j.

Afin d'améliorer l'efficacité de l'algorithme, des optimisations ont été réalisées au niveau de la gestion des téléchargements. Les pages déjà visitées par les fourmis sont stockées au niveau de chaque nid. Ainsi, lorsqu'une fourmi visite un document déjà pris en compte par une fourmi du même nid, la demande de téléchargement est interceptée par l'algorithme de gestion du nid et l'évaluation de la page - stockée au préalable - est retournée directement à la fourmi. Les nids étant indépendants entre eux, deux fourmis appartenant à deux nids différents et voulant visiter la même page Web effectueront deux demandes aux ressources de téléchargement.

Au fur et à mesure du déroulement de l'algorithme, les meilleures pages sont sélectionnées, de manière transparente pour les nids, au niveau des ressources de téléchargement dans une population de documents. De cette manière, l'algorithme *Antsearch* n'a pas à se préoccuper de la gestion des meilleures pages rencontrées et simplifie l'algorithme de gestion des nids. Ce dernier ne se préoccupe ainsi que de guider la recherche

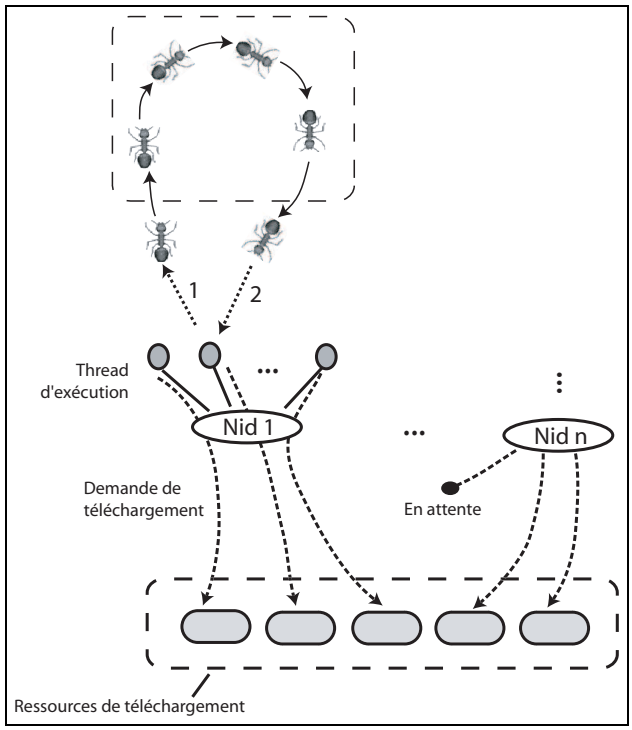


FIG. 6.1 – Modélisation de la parallélisation de *Antsearch* sur deux niveaux : les threads d'exécution de la stratégie de recherche d'une part et les ressources de téléchargement d'autre part.

des fourmis.

Lorsque la recherche est terminée, les résultats correspondent uniquement aux documents présents dans cette population. Elle ne contient que les meilleures pages rencontrées au cours de la recherche par toutes les fourmis de toutes les colonies mises en œuvre.

Le critère d'arrêt utilisé ici peut être le même que celui utilisé dans l'algorithme *API* :

- si s^+ n'a pas été amélioré depuis un certain nombre d'itération ou d'un certain temps,
- un nombre maximum d'itérations a été atteint,
- un certain nombre d'évaluations de solutions de f a été atteint.

Nous retiendrons ce dernier critère car il est similaire à celui utilisé dans *Geniminer* et *GeniminerII* présentés respectivement dans les chapitres 4 et 5. En effet, le nombre d'évaluations correspond au nombre de téléchargements effectués par l'algorithme. De cette manière, il est possible de comparer équitablement ces algorithmes entre eux.

On peut signaler que le système de cache introduit dans la gestion des nids entraîne quelques adaptations de ce critère. En effet, lorsqu'une page demandée en téléchargement par une fourmi est stockée dans un nid, il n'y a pas de nouveau téléchargement ni de nouvelle évaluation effectuée. Ces éléments sont déjà présents au sein du nid et

(1) **si** $m_{f_i} < p$ **alors**
(2) /* La mémoire de la fourmi n'est pas pleine */
(3) Construction d'un site de chasse autour du nid : $S_{f_i} \leftarrow O_{explo}(N_n, A_{site})$
(4) $m_{f_i} \leftarrow m_{f_i} + 1$
(5) Initialisation du compteur d'échecs du site construit : $e_{i,j} \leftarrow 0$
(6) **sinon**
(7) Soit s_j le site que la fourmi a exploré à sa dernière sortie
(8) **si** $e_{i,j} > 0$ **alors**
(9) /* la dernière exploration de s_j a été infructueuse */
(10) Choisir un site de chasse s_j ($j \in 1, \ldots, p$) contenu dans S_{f_i}. Ce choix se fait en fonction d'une probabilité croissante avec le degré de pertinence du site à la requête[1].
(11) **finsi**
(12) Exploration locale autour du site s_j : $s' \leftarrow O_{explo}(s_j, A_{local})$
(13) **si** $f(s') < f(s_j)$ **alors**
(14) $s_j \leftarrow s'$
(15) $e_{i,j} \leftarrow 0$
(16) **sinon**
(17) $e_{i,j} \leftarrow e_{i,j} + 1$
(18) **si** $e_{i,j} > P_{local}$ **alors**
(19) Effacer le site s_j de la mémoire de la fourmi
(20) $m_{f_i} \leftarrow m_{f_i} - 1$
(21) **finsi**
(22) **finsi**
(23) **finsi**

ALG 6.2: FOURRAGEMENT(f_i,N_n) : description du comportement local d'une fourmi.

sont, par conséquent, renvoyés directement à la fourmi concernée. Puisqu'il ne s'agit pas d'un réel téléchargement mais d'une gestion des données interne à l'algorithme, nous ne comptabilisons naturellement pas cette demande parmi les demandes impliquant l'arrêt de l'algorithme.

6.2.3 Étude des paramètres

Cet algorithme a la particularité de posséder un grand nombre de paramètres résumés dans le tableau 6.1. Il est par conséquent très difficile de déterminer les interactions croisées qui peuvent exister parmi toutes les combinaisons possibles de paramètres que nous pouvons générer. Nous allons toutefois tenter de dégager dans cette section quelques constatations que nous avons pu observer dans les nombreux tests que nous

[1]Par exemple, avec 3 sites de chasse en mémoire, le premier sera choisi selon une probabilité de $\frac{3}{6}$, le deuxième selon une probabilité de $\frac{2}{6}$ et le dernier selon une probabilité de $\frac{1}{6}$.

Paramètre	Description
N_{nids}	Nombre de nids
$N_{fourmis}$	Nombre de fourmis par nids
A_{site}	Amplitude de création d'un site de chasse
A_{local}	Amplitude d'exploration locale
P_{local}	Patience locale d'une fourmi
p	Nombre de sites mémorisés par une fourmi
$MaxDepFou$	Nombre de mouvements maximum des fourmis d'un nid avant son déplacement
P_{creat}	Probabilité d'utiliser l'opérateur O_{rand} pour déterminer la nouvelle position d'un nid
N_{thread}	Nombre de threads d'exécution par nid
$N_{download}$	Nombre de téléchargements maximum simultanés

TAB. 6.1 – Récapitulatif de l'ensemble des paramètres de l'algorithme *Antsearch*.

avons effectués.

Cependant, certaines contraintes de paramètres ont pu être remarquées lors de l'analyse du comportement des fourmis réelles. Ces contraintes ont été élaborées et testées pendant des millions d'années en ce qui concerne les fourmis. Il est donc naturel et raisonnable de les considérer, et d'essayer d'établir un parallèle avec notre modélisation. De plus, le nombre d'exécutions concurrentes de l'algorithme de fourragement ainsi que le nombre de téléchargements effectués dépendent de l'architecture utilisée pour les tests et nous ne ferons par conséquent pas varier ces deux paramètres dans les évaluations de cette méthode.

La première association que nous pouvons effectuer concerne les paramètres A_{site} et A_{local} gérant respectivement l'amplitude de création d'un site de chasse autour du nid des fourmis et l'amplitude d'exploration locale autour d'un site de chasse. Ces deux valeurs sont en effet corrélées dans la nature chez les fourmis réelles. Nous avons, par conséquent, choisi de faire varier ces paramètres de manière proportionnelle d'un facteur 2. En effet, on a constaté dans la section 1.2.1 que le diamètre d'Internet était de seulement une vingtaine de liens. Par conséquent, plus une fourmi explore de liens à la recherche d'une proie, plus sa probabilité de ne pas trouver de documents intéressants augmente (le nombre de documents accessibles et donc possiblement parasites augmente très vite). Chez les fourmis réelles, l'amplitude de création d'un site de chasse est plus importante que l'amplitude d'exploration locale. Nous avons donc décidé de conserver cette caractéristique.

Les paramètres N_{nids} et $N_{fourmis}$ gérant le nombre de nids et le nombre de fourmis par nid fixe le nombre total de fourmis mises en œuvre pour explorer l'espace de recherche. Avec ces deux paramètres, on cherche à savoir s'il est plus judicieux d'explorer intensivement un endroit particulier de l'espace de recherche ou au contraire s'il vaut mieux répartir les fourmis sur plusieurs sites. Le nombre d'explorations réalisées par l'ensemble des fourmis de la colonie déterminant l'arrêt de la recherche, nous avons dans tous les cas considéré le même nombre de fourmis à chaque exécution de l'algo-

Paramètre	**Valeurs**
A_{site}/A_{local}	{2/1, 4/2, 6/3}
P_{creat}	{0.5, 0.8}
$N_{nids}/N_{fourmis}$ /$MaxDepFou$	{1/80/400, 10/8/40}
p	{1, 3, 5}

TAB. 6.2 – Valeur des paramètres testés.

rithme. On peut également lier un autre paramètre aux deux précédents : $MaxDepFou$ qui représente le nombre de mouvements effectués par l'ensemble des fourmis d'un nid avant de demander le déplacement de ce nid. Ici aussi, dans un souci d'égalité, chaque fourmi effectuera le même nombre de pas en moyenne.

Nous avons également décidé de fixer la patience locale de chaque fourmi (paramètre P_{local}) car l'étude de l'algorithme API a montré qu'il fallait que cette valeur soit assez faible dans tous les cas de figure pour obtenir les meilleurs résultats.

Le tableau 6.2 indique les différentes combinaisons de paramètres que nous avons utilisées pour effectuer l'ensemble de nos tests.

Pour chaque combinaison de paramètres, nous avons effectué 3 tests afin d'obtenir une moyenne de résultats sur plusieurs exécutions de l'algorithme. Cela représente un total de 36×3 exécutions pour chaque requête testée. La mesure de la qualité des résultats est la même que celle utilisée pour l'algorithme *GeniminerII* dans le chapitre 5. Cette mesure est présentée dans la section 5.3.2 et pour chaque test, nous avons évalué la population de résultats obtenus sous trois angles différents : en analysant les 30 documents les plus pertinents, puis les 60 premiers et enfin les 100 premiers.

Nous avons tenté de mesurer l'influence de chaque paramètre sur les performances de l'algorithme en fixant un paramètre et en faisant varier l'ensemble des autres paramètres. Les tableaux 6.3, 6.4, 6.5 et 6.6 représentent les résultats obtenus respectivement sur les requêtes 1, 6, 7 et 10 de la table 5.1 page 138.

Cette méthode d'évaluation présente un inconvénient : l'utilisation de la moyenne des résultats sur tous les jeux de paramètres peut cacher un certain nombre d'associations entre les paramètres qui contrediraient les tendances constatées. Cependant elle permet de conclure sur l'efficacité globale de certaines configurations et de tirer quelques conclusions quant au paramétrage de l'algorithme.

En prenant en considération les résultats obtenus pour le premier paramètre testé (la combinaison des variables A_{site} et A_{local}), on constate que sur chacune des requêtes présentées, la configuration d'exploration locale étendue (valeurs respectives de 6 et de 3 pour les deux variables) ne mène quasiment jamais à l'obtention des meilleurs résultats. La seule exception concerne la requête 7 (tableau 6.5) en prenant compte les 60 premiers résultats dans l'évaluation. Cependant, on remarque que pour cette requête, quelle que soit la configuration choisie, les résultats obtenus sont sensiblement identiques et ne permettent pas réellement de conclure sur le meilleur paramètre à choisir. Dans

A_{site}/A_{local}	**30**	**60**	**100**
2/1	80.41	82.89 ▶	88.31
4/2	▶ 81.14	▶ 83.96	88
6/3	76.87	78.89	83.93

P_{creat}	**30**	**60**	**100**
0.5	▶ 79.64	▶ 82.3	86.1
0.8	79.3	81.52 ▶	87.39

$N_{nids}/N_{fourmis}$ /$MaxDepFou$	**30**	**60**	**100**
1/80/400	▶ 79.58	81.64	86.36
10/8/40	79.36	▶ 82.19	▶ 87.12

p	**30**	**60**	**100**
1	77.23	81.29	86.38
3	80.37	▶ 82.36	▶ 87.12
5	▶ 80.81	82.1	86.74

TAB. 6.3 – Évolution de la qualité des 30, 60 et 100 premiers résultats d'*Antsearch* obtenus au bout de 1000 téléchargements en fonction des paramètres de l'algorithme sur la requête 1.

A_{site}/A_{local}	**30**	**60**	**100**
2/1	▶ 78.54	▶ 83.44	▶ 85.40
4/2	77.44	81.56	84.28
6/3	76.11	78.47	81.37

P_{creat}	**30**	**60**	**100**
0.5	75.66	80.23	82.84
0.8	▶ 79.07	▶ 82.08	▶ 84.54

$N_{nids}/N_{fourmis}$ /$MaxDepFou$	**30**	**60**	**100**
1/80/400	71.26	75.77	77.99
10/8/40	▶ 83.46	▶ 86.55	▶ 89.38

p	**30**	**60**	**100**
1	77.10	80.81	83.32
3	▶ 77.75	▶ 81.49	▶ 85.21
5	77.24	81.18	82.53

TAB. 6.4 – Évolution de la qualité des 30, 60 et 100 premiers résultats d'*Antsearch* obtenus au bout de 1000 téléchargements en fonction des paramètres de l'algorithme sur la requête 6.

A_{site}/A_{local}	**30**	**60**	**100**
2/1	82.76	84.25	83.41
4/2	▶ 83.96	84.09	▶ 83.61
6/3	82.76	▶ 85.00	83.19

P_{creat}	**30**	**60**	**100**
0.5	82.80	84.32	83.16
0.8	▶ 83.51	▶ 84.57	▶ 83.64

$N_{nids}/N_{fourmis}$ /$MaxDepFou$	**30**	**60**	**100**
1/80/400	▶ 86.69	▶ 87.15	▶ 86.79
10/8/40	79.62	81.74	80.01

p	**30**	**60**	**100**
1	78.24	80.74	80.11
3	84.57	84.56	83.21
5	▶ 86.66	▶ 88.04	▶ 86.88

TAB. 6.5 – Évolution de la qualité des 30, 60 et 100 premiers résultats d'*Antsearch* obtenus au bout de 1000 téléchargements en fonction des paramètres de l'algorithme sur la requête 7.

A_{site}/A_{local}	**30**	**60**	**100**
2/1	▶ 73.61	▶ 73.22	▶ 79.09
4/2	70.69	68.91	72.54
6/3	69.12	70.12	73.48

P_{creat}	**30**	**60**	**100**
0.5	▶ 75.66	▶ 80.23	▶ 82.84
0.8	71.21	69.61	74.35

$N_{nids}/N_{fourmis}$ /$MaxDepFou$	**30**	**60**	**100**
1/80/400	▶ 73.39	▶ 73.10	▶ 77.65
10/8/40	68.90	68.40	72.43

p	**30**	**60**	**100**
1	▶ 73.22	▶ 72.80	▶ 77.96
3	70.39	70.16	73.29
5	69.81	69.29	73.86

TAB. 6.6 – Évolution de la qualité des 30, 60 et 100 premiers résultats d'*Antsearch* obtenus au bout de 1000 téléchargements en fonction des paramètres de l'algorithme sur la requête 10.

les autres requêtes, les deux autres configurations permettent d'obtenir des résultats plus pertinents. Sur la requête 1 (tableau 6.3), ces deux couples de valeurs obtiennent des résultats similaires. Il en est de même pour la requête 6 (tableau 6.4) en notant cependant une légère dominance de la plus faible exploration locale (valeurs 2 et 1 pour A_{site} et A_{local}). Cette dominance est confirmée dans la requête 10 (tableau 6.4) pour laquelle ce couple de valeurs obtient une nette avance sur les autres configurations.

Le deuxième paramètre (P_{create}) gère la probabilité d'utiliser l'opérateur O_{rand} pour déterminer la nouvelle position d'un nid. Les deux valeurs testées (0.5 et 0.8) mènent à des résultats similaires sur les requêtes 1 et 7. Sur cette dernière requête, une valeur de 0.8 apporte les meilleurs résultats quel que soit le nombre de documents pris en compte dans l'évaluation mais l'écart entre les résultats reste très minime (inférieur à 1% du meilleur résultat). Les deux autres requêtes voient les dominances s'inverser : la valeur 0.8 est meilleure sur la requête 6 et moins forte sur la requête 10. Cependant l'écart entre les résultats obtenus pour les deux valeurs de paramètre est beaucoup plus important sur cette dernière requête. Il semble par conséquent naturel de fixer la valeur de ce paramètre à 0.5.

Les paramètres suivants, gérant la dispersion des fourmis sur l'espace de recherche sont plus délicats à déterminer. En effet, alors que les résultats obtenus sont relativement similaires pour la requête 1 - bien que l'on constate une légère dominance de la configuration employant plusieurs nids - les autres requêtes montrent une différence importante pour les deux configurations. L'utilisation d'un unique nid apporte de meilleurs résultats sur les requêtes 7 et 10 avec une différence d'évaluation allant de 5 à 7 points. La situation est inversée sur la requête 6 et la différence d'évaluation atteint 11 points dans le meilleur des cas et plus de 12 points dans le pire des cas. Il semble que l'utilisation d'un nid unique soit à double tranchant : soit l'algorithme parvient à déterminer une zone de l'espace de recherche comportant beaucoup de documents pertinents, soit peu de pages intéressantes figureront dans les résultats. Ce comportement nous pousse à choisir d'utiliser plusieurs nids pour obtenir des résultats les plus homogènes possibles.

Le dernier paramètre détermine le nombre de sites de chasse mémorisables par une fourmi. Une fois de plus, les résultats obtenus sont similaires quelle que soit la configuration choisie pour la requête 1. Il en est de même pour la requête 6 excepté lorsque l'on prend en compte 100 documents résultats dans l'évaluation de la méthode. Dans ce cas, la mémorisation de 3 sites permet d'obtenir les meilleurs résultats. Les deux autres requêtes voient tour à tour dominer les valeurs 1 et 5 du paramètre alors que la valeur 3 se place dans les deux cas en position médiane. Pour les mêmes raisons qui nous ont fait préférer une exploration plus diversifiée de l'espace de recherche, nous choisirons dans la suite d'utiliser une taille de mémoire d'une fourmi pouvant contenir 3 sites de chasse.

Les analyses que nous avons faites sur la méthode *Antsearch* ne prennent pas en compte les relations privilégiées qui peuvent exister entre des couples ou des trio de paramètres. Cependant elles permettent de définir des valeurs offrant un bon compromis pour l'ensemble des requêtes testées. Ainsi dans la suite de ce rapport, nous fixerons les valeurs de ces paramètres comme suit :

- A_{site}/A_{local} : 2/1
- P_{create} : 0.5
- $N_{nids}/N_{fourmis}/MaxDepFou$: 10/8/40
- p : 3

La suite de ce chapitre est dévolue à la création d'une autre heuristique de recherche d'information. Elle se base sur un algorithme tabou. Cette méthode a obtenu en optimisation de très bons résultats. Il est par conséquent intéressant d'observer son comportement sur notre problème.

6.3 Heuristique basée sur un algorithme tabou

Dans cette section nous présentons une adaptation d'une méthode de recherche tabou de façon à rendre applicable cette modélisation à la recherche de documents sur Internet. La méthode tabou est utilisée en optimisation afin de converger rapidement vers les extrema locaux d'un espace de recherche. Nous avons détaillé le fonctionnement global de ce type d'algorithme dans la section 2.2 du chapitre 2.

Il est intéressant de comparer les heuristiques que nous avons précédemment formulées à une telle méthode car il s'agit d'une méthode d'exploration locale ayant montré de bonnes aptitudes à la recherche rapide de *bonnes* solutions à un problème d'optimisation.

6.3.1 Description algorithmique

Dans un algorithme tabou, on choisit d'explorer le meilleur voisin (ou un meilleur voisin selon le cas) à chaque itération à partir du point courant. Ce nouveau point doit correspondre à une solution non tabou, c'est-à-dire ne doit pas figurer dans la liste tabou gérée par l'algorithme. Il est ensuite placé à son tour dans la liste tabou. La taille de cette liste est bornée à un nombre maximum et s'il ne reste plus de place libre pour insérer le nouvel élément, on élimine la plus ancienne solution trouvée depuis le début de la recherche. Cette liste est, par conséquent, gérée en FIFO. Elle a été mise en place dans le but de ne pas *boucler* la recherche dans une zone de l'espace et de permettre de *s'échapper* d'un optimum local. Les extrema locaux ayant déjà été visités se retrouvent dans la liste tabou et ne sont, par conséquent, plus explorés.

Les meilleures solutions rencontrées au cours de la recherche sont stockées dans une population de résultats similaires à celles utilisées dans *GeniminerII* dans le chapitre 5 et dans *Antsearch* décrit dans la section 6.2 de ce chapitre.

Le critère d'arrêt utilisé dans une recherche tabou peut être de différentes natures :
- après un nombre fixé d'itérations,
- après un nombre fixé d'étapes n'ayant pas amélioré la solution.
- après qu'un certain temps se soit écoulé sans que la meilleure solution n'ait été améliorée.

Afin de rester conforme à ce que nous avons effectué pour les algorithmes précédents et afin de faciliter la comparaison des méthodes entre elles, nous avons opté pour un

arrêt au bout d'un certain nombre d'itérations. Ce nombre est directement corrélé avec le nombre de téléchargement effectués par l'algorithme ce qui est similaire à ce que nous avons réalisé dans les autres méta-heuristiques.

Une fois la notion de voisin fixée, une méthode tabou n'est paramétrée que par une seule variable : la taille de la liste tabou. La méthode *tabou* peut donc être vue comme une généralisation de la recherche d'optimum local (si la taille de la liste est nulle, on retombe dans le cas du *hill-climbing*).

6.3.1.1 Adaptation pratique à la recherche de documents sur Internet

La transformation de la méthode tabou dans notre problématique consiste à parcourir le Web de pages en pages à la recherche de documents pertinents en suivant les liens hypertextes les reliant. La fonction d'évaluation indispensable à la recherche pour déterminer la valeur de la solution trouvée est en réalité une évaluation de la pertinence de la page par rapport à la requête de l'utilisateur. Nous utilisons ici la même fonction d'évaluation que celle utilisée dans l'algorithme *Antsearch* décrite dans la section 3.3.2.

L'algorithme 6.3 décrit la modélisation tabou adaptée à la problématique de recherche d'information sur Internet. L'initialisation d'un point de l'espace de recherche se fait par l'intermédiaire de l'opérateur O_{rand} qui interroge les moteurs de recherche classiques et évalue un lien donné en résultat. Et d'une manière similaire aux méta-heuristiques présentées précédemment, la recherche d'un voisin d'un point S dans l'espace de recherche est réalisée au moyen de l'opérateur O_{explo}. L'exploration locale est effectuée en prenant en compte les liens directs du point exploré, c'est-à-dire les documents à une distance de 1 lien. Nous avions utilisé ce même principe dans les algorithmes *Geniminer* et *GeniminerII*.

Comme nous l'avons signalé précédemment, l'arrêt de la recherche s'effectue de manière automatique lorsqu'un certain nombre de pages Web ont étés traitées. Cependant, il existe d'autres causes menant à la fin de la recherche. Celle-ci peut en effet prendre fin s'il n'y a plus de page Web issues des moteurs de recherche à explorer.

La recherche de voisin en voisin s'arrête lorsqu'on se trouve face à une page *stérile*, c'est-à-dire sans liens sortants non tabous. Dans ce cas, la recherche reprend typiquement à partir d'une page issue d'un moteur de recherche par l'intermédiaire de l'opérateur O_{rand}. D'autres solutions ont été envisagées mais nous ont conduit à quelques problèmes. Nous avions imaginé, dans un tel cas, remonter à la page mère dont était issue la page stérile et rechercher à partir de ce point un autre chemin par l'intermédiaire d'un voisin non tabou. Cependant, la seule possibilité pour réinterroger un moteur de recherche aurait été d'avoir exploré tous les liens à notre disposition sur le chemin parcouru. Or, nous avons vu dans la section 1.2.1 du chapitre 1 qu'Internet est très fortement connecté. Nous aurions parcouru dans ce cas une grande partie de ce réseau avant d'avoir épuisé tous les liens à notre disposition. Dans le cas du Web, cette solution est par conséquent inenvisageable.

```
(1)  StopAlgo ← faux
(2)  /* Initialiser S à partir d'une page extraite d'un moteur de recherche */
(3)  S ← O_rand
(4)  Ajouter S à la liste tabou
(5)  tantque (StopAlgo = faux) ∧ la condition d'arrêt n'est pas vérifiée faire
(6)     répéter
(7)        S' ← O_explo(S, 1)
(8)     jusqu'à (S' ∉ liste tabou) ∨ (∄ un voisin non tabou de S) fin
(9)     si ∄ un voisin non tabou de S alors
(10)       si ∄ de pages issues de moteurs de recherche alors
(11)          StopAlgo ← vrai
(12)       sinon
(13)          S ← O_rand
(14)       finsi
(15)    finsi
(16)    si la liste tabou est pleine alors
(17)       Remplacer le dernier élément de la liste tabou par la solution S'
(18)    sinon
(19)       Ajouter S' à la liste tabou
(20)    finsi
(21)    S ← S'
(22) fintantque
```

Alg 6.3: Algorithme principal de *Tabusearch*

Afin de déterminer un ensemble de pages pertinentes, et non pas *le* document le plus pertinent à la requête de l'utilisateur, nous avons, de la même manière que pour l'algorithme *Antsearch*, utilisé une population de résultats gérée de manière transparente par rapport à l'algorithme tabou. Les meilleurs documents sont stockés par le module de téléchargement dans une population au fur et à mesure des demandes d'évaluation de pages Web par l'algorithme tabou. Lorsque l'exécution de l'algorithme est terminée, la population de pages les plus pertinentes est ainsi retournée à l'utilisateur.

Un des problèmes que nous pouvons rencontrer en utilisant un tel algorithme de recherche tabou est de ne pas utiliser suffisamment l'opérateur O_{rand} afin d'explorer de nouveaux points de l'espace de recherche. En effet, le graphe représentant Internet est très fortement interconnecté. Par conséquent, le suivi de liens hypertextes ne mène que rarement à un ensemble de pages *stériles*. Par conséquent, nous avons décidé de paralléliser la recherche en exécutant plusieurs listes tabou simultanément.

Afin de ne pas laisser deux threads d'exécution explorer la même zone de l'espace de recherche, nous avons décidé d'utiliser une liste tabou commune à l'ensemble des algorithmes de recherche tabou. En effet, il semble évident que si deux recherches tabou

arrivent sur la même page, elles effectueront la même exploration et donc donneront toutes deux le même résultat, ce qui ne nous intéresse pas. Cette liste tabou générale contient toutes les pages déjà fournies par l'opérateur O_{rand} et donc issues des moteurs de recherche classiques. Ce choix nous a obligé à mettre en œuvre un accès protégé à la liste tabou générale par un mécanisme de verrou utilisant des sémaphores.

On peut également signaler que compte tenu du fait que la parallélisation est gérée par le système d'exploitation - les threads sont réellement mis en concurrence en temps partagé -, il est impossible de prédire le nombre d'itérations effectuées par chaque entité parallélisée durant une recherche. C'est-à-dire que le nombre d'évaluations accordé à chaque algorithme tabou sera aléatoire puisque accordé par le système d'exploitation indépendamment les uns des autres. Le critère d'arrêt est donc basé sur le nombre d'évaluations réalisé par l'ensemble des threads.

6.3.2 Expérimentations

Afin de tester l'efficacité de cette méthode, il convient d'examiner les paramètres de l'algorithme afin d'exploiter au mieux ses ressources. Dans la recherche tabou, il n'existe pas beaucoup de paramètres et le plus classique représente la taille de la liste tabou employée. Mais dans l'approche parallèle que nous avons adoptée, un autre paramètre intervient : le nombre de listes tabou utilisées simultanément.

Nous avons testé ces paramètres en utilisant la même méthodologie utilisée pour évaluer les paramètres de *GeniminerII* (voir chapitre 5) et de *Antsearch* (voir section 6.2). Nous avons pour cela employé la fonction d'évaluation multicritère définit dans la section 3.3.2 du chapitre 3. Elle permet de mesurer efficacement la qualité des différents documents sélectionnés.

Dans la suite, nous avons sélectionné quatre requêtes représentatives des différents comportements de l'algorithme que nous avons observés. Il s'agit des requêtes 1, 6, 7 et 10 présentées dans le tableau 5.1 page 138. Pour chacune d'entre elles, nous avons mesuré la qualité des résultats obtenus au bout de 1000 téléchargements, en effectuant une moyenne sur trois tests. En effet, bien que la méthode tabou employée ici suive toujours le même chemin, nous obtenons des résultats légèrement différents sur plusieurs exécutions de l'algorithme. Ceci est dû au type de parallélisation employée. C'est en effet le système d'exploitation qui gère l'exécution simultanée des différentes listes tabou et deux listes peuvent donc effectuer un nombre de mouvements différents.

La méthode d'évaluation de la qualité d'un résultat est la même que celle employée dans les tests réalisés sur *GeniminerII* et *Antsearch*. Elle est décrite dans la section 5.3.2 du chapitre 5. Elle permet de mesurer facilement la qualité globale des résultats d'une méthode de recherche. Plus grande est la valeur accordée à un résultat, meilleur est ce dernier.

Les paramètres ont été échantillonnés sur plusieurs valeurs : 6 tailles de listes tabou (5, 10, 20, 30, 40, 50) et 4 nombres de listes employées simultanément (10, 15, 20, 25). Les tests ont été effectués en faisant varier conjointement ces deux paramètres.

La figure 6.2 représente les résultats obtenus sur la première requête de test. La figure *a)* présente les moyennes obtenues sur les 30 premiers individus de la population de résultats, la figure *b)* ceux obtenus sur les 60 premiers résultats et la figure *c)* ceux obtenus sur les 100 premiers résultats.

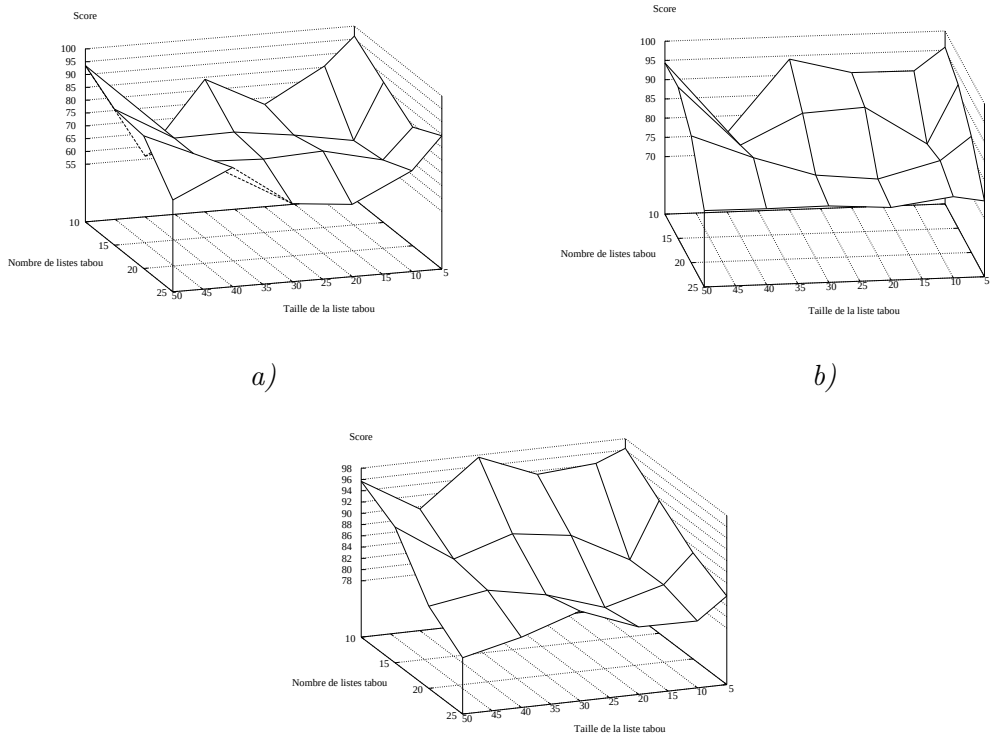


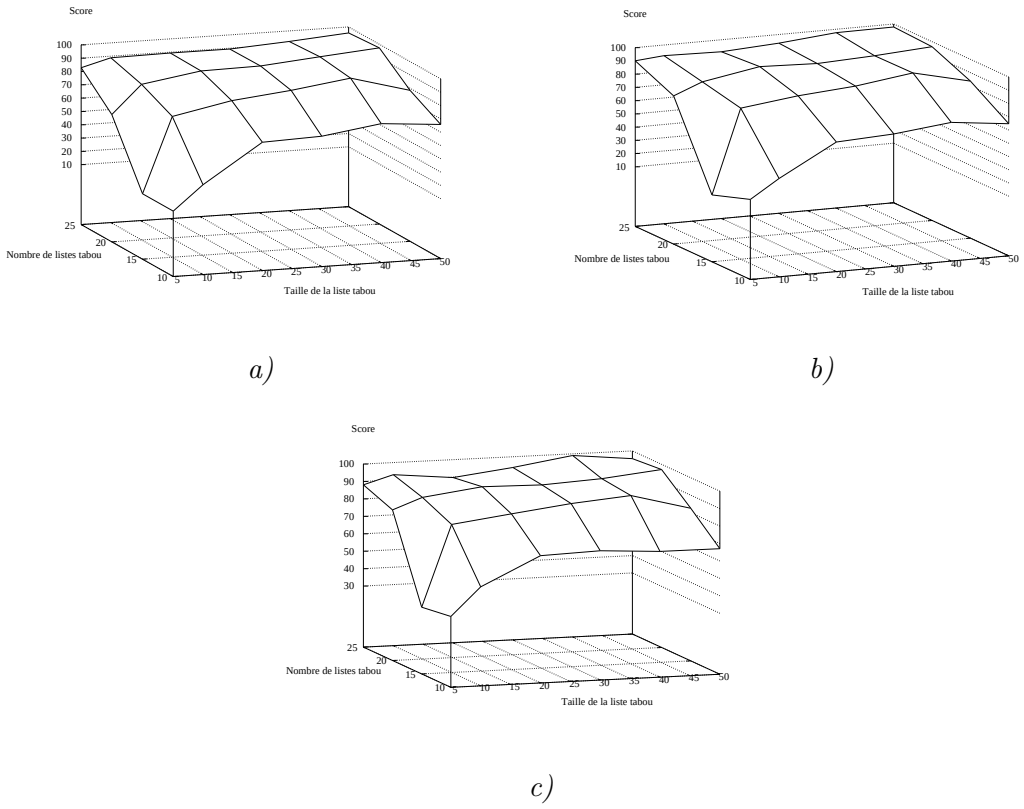
FIG. 6.2 – Influence conjointe de la taille d'une liste tabou et du nombre de listes employées en parallèle pour l'algorithme *Tabusearch* pour la requête 1 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

Dans la courbe *a)*, on remarque que les meilleurs résultats sont obtenus pour de très faibles et grandes valeurs de la taille des listes tabou. Le nombre de listes employées en parallèle n'influence que relativement peu la qualité des résultats. On note toutefois une légère diminution de cette qualité avec un nombre grandissant de listes. Cette différence s'amplifie si l'on prend en compte un plus grand nombre d'individus dans l'évaluation des résultats.

Alors que la qualité des premiers individus de la population dépend en priorité de la taille des listes tabou, on remarque dans les figures *b)* et surtout *c)*, que le nombre de listes tabou utilisées simultanément prend une grande importance dans la qualité globale des résultats. Cependant, comme nous allons le voir dans la suite de ce chapitre,

il ne s'agit pas d'une généralité et nous n'avons observé ce comportement que sur très peu de requêtes.

La figure 6.3 montre le même type de résultats obtenus sur la requête 6 de la table 5.1. La figure *a)* correspond aux 30 premiers individus de la population de résultats, la figure *b)* aux 60 premiers résultats et la figure *c)* aux 100 premiers résultats.



a) *b)*

c)

FIG. 6.3 – Influence conjointe de la taille d'une liste tabou et du nombre de listes employées en parallèle pour l'algorithme *Tabusearch* pour la requête 6 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

Ces courbes montrent une tendance bien différente de celle observée sur la précédente requête. La qualité des résultats est comparable dans les trois cas. Que l'on prenne en compte uniquement les premiers individus résultats ou un plus grand nombre, les courbes sont similaires. Elles montrent toutes une amélioration de la qualité des résultats obtenus avec une augmentation de la taille des listes tabou et du nombre de listes employées simultanément. Cette progression, n'est pas linéaire : plus le nombre de listes employées simultanément est faible et plus la taille des listes tabou est petite, plus rapide est la baisse de qualité observée dans la population de résultats.

Cette situation peut s'expliquer par une simple remarque. Les résultats renvoyés

par les moteurs de recherche sur cette requête correspondent souvent à des pages Web comportant beaucoup de liens hypertextes internes à un site. Il est donc fréquent de rencontrer des liens pointant sur une page déjà visitée par l'algorithme. Dans le cas d'une taille de liste limitée, l'algorithme oubli rapidement qu'il a examiné une page particulière. La sélection des liens le guide donc de manière récurrente vers des documents analysés précédemment, ce qui fait perdre du temps - et par conséquent de l'efficacité - à la méthode. De plus, la présence importante de liens hypertextes internes dans les résultats issus des moteurs de recherche n'aide pas à diversifier l'espace exploré par la méthode tabou. L'utilisation d'un grand nombre de listes tabou initialise l'algorithme sur un plus grand nombre de points de l'espace de recherche facilitant la recherche de documents pertinents.

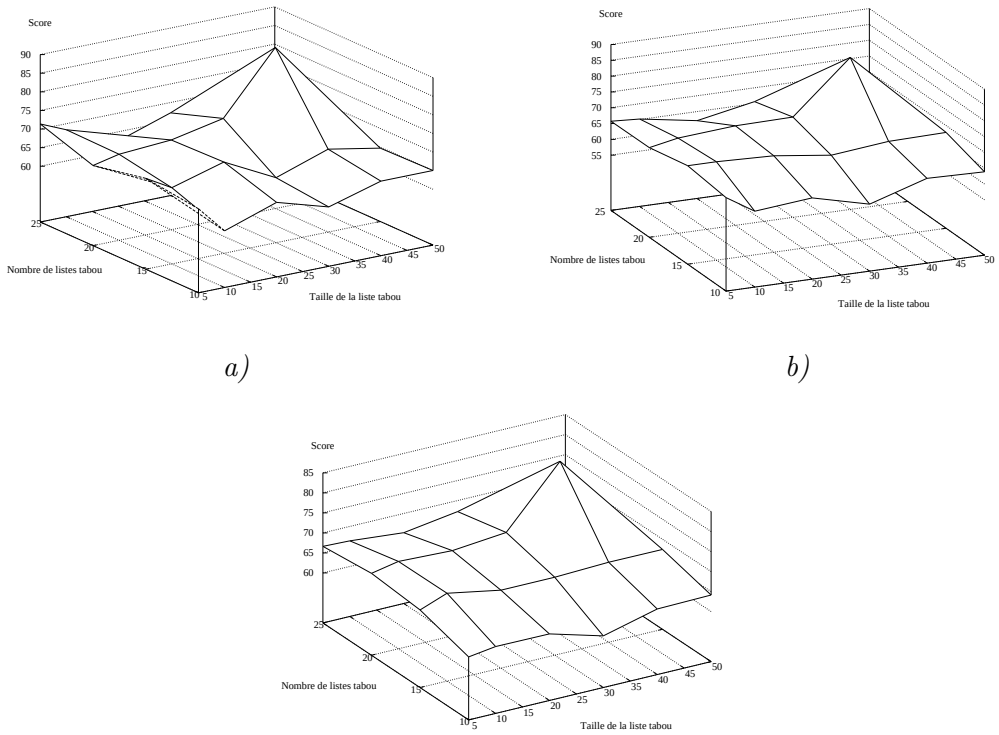


FIG. 6.4 – Influence conjointe de la taille d'une liste tabou et du nombre de listes employées en parallèle pour l'algorithme *Tabusearch* pour la requête 7 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

La figure 6.4 montre l'évolution de la qualité des résultats de la septième requête en fonction des paramètres testés.

Ces courbes ne montrent que peu de variations en comparaison avec celles précédem-

ment analysées. Les plus fortes variations sont visibles lorsque l'on considère uniquement les 30 premiers individus de la population de résultats sur la figure *a)*. La tendance montre qu'un grand nombre de listes tabou en concurrence améliore les résultats. Ce paramètre en combinaison avec une liste tabou de taille élevée donne les meilleurs résultats sur cette requête. On peut signaler également sur cette courbe qu'une faible taille de liste mène également à de bons résultats, mais dans de moindres proportions.

Les courbes *b)* et *c)*, prenant en compte respectivement les 60 et les 100 premiers résultats obtenus par la méthode tabou, confirment la dominance d'une combinaison comprenant beaucoup de listes tabou de grande taille.

Cette requête traite d'un sujet général dont les résultats pertinents sont présents en abondance. Un indexeur a donc de fortes chances de recenser des documents intéressants. La méthode prenant en compte le plus de résultats possible issus d'un moteur de recherche - et donc d'un indexeur - correspond ainsi à celle ayant les meilleurs résultats. Cette situation est amplifiée en prenant en compte un grand nombre d'éléments, ce qui explique que la courbe se soit lissée en comparant les figures *a)* et *c)*.

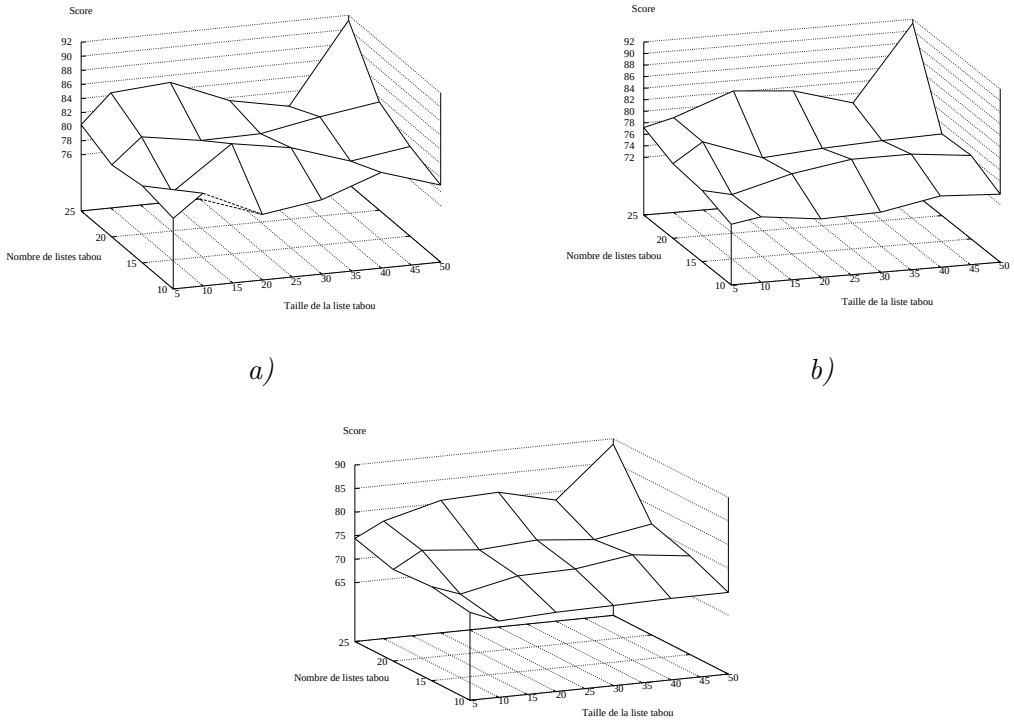


FIG. 6.5 – Influence conjointe de la taille d'une liste tabou et du nombre de listes employées en parallèle pour l'algorithme *Tabusearch* pour la requête 8 sur les *a)* 30, *b)* 60, *c)* 100 premiers résultats obtenus au bout de 1000 téléchargements.

La figure 6.5 représente une situation relativement similaire. Il s'agit des résultats obtenus en analysant la requête 10.

On observe sur les courbes une légère pente favorisant l'utilisation d'un grand nombre de listes tabou. Une fois de plus, les meilleurs résultats (observables sur la figure *a)*) peuvent être obtenus en utilisant une faible ou une grande taille de liste. Lorsque l'on prend en compte un plus grand nombre de résultats (figures *b)* et *c)*), la combinaison regroupant un grand nombre de listes de grande taille permet d'obtenir les documents les plus pertinents.

Les résultats obtenus sur cette requête sont relativement représentatifs d'un grand nombre de cas que nous avons observés et confirme les relations existant entre qualité des résultats et nombre de listes tabou mises en œuvre.

Sur la majorité des requêtes que nous avons examinées, on note une tendance à accroître la qualité des résultats avec un nombre important de listes tabou s'exécutant de manière concurrente. En comparaison à ce paramètre, la taille de la liste tabou ne semble avoir qu'un impact limité. Toutefois on observe qu'une trop petite taille a tendance à handicaper la recherche et à produire des résultats de moindre qualité.

Sur la base de ces remarques, nous avons décidé d'utiliser dans la suite de nos tests 20 listes tabou d'une quarantaine d'individus en concurrence. Ce paramètre semble être un bon compromis évitant d'obtenir des résultats de trop mauvaise qualité.

6.4 Conclusion

Ce chapitre nous a permis de démontrer que plusieurs types d'algorithmes de nature très différentes peuvent utiliser le modèle que nous avons construit. Les deux méta-heuristiques que nous avons élaborées ont connu chacune un certain succès dans le monde de l'optimisation combinatoire. Il était donc intéressant de les appliquer à notre problème.

L'algorithme *Antsearch* a l'inconvénient de posséder un grand nombre de paramètres. Il est, par conséquent, difficile de mesurer les interactions combinées de ces éléments. La solution que nous avons adoptée consiste à fixer un paramètre en faisant varier l'ensemble des autres éléments. Cela nous a permis de conclure sur l'efficacité globale de chaque paramètre.

L'algorithme *Tabusearch* est basé sur un algorithme tabou. Le nombre de paramètres gérant son comportement est donc limité et il est beaucoup plus aisé de fixer les meilleures valeurs possibles pour chacun d'eux.

Après avoir défini trois algorithmes de recherche s'inspirant de méthodologies bien différentes, il convient de les évaluer ensemble afin de déterminer la meilleure approche pour notre problème. Le chapitre suivant se consacre à cette étude et soumet la méta-heuristique apportant les meilleurs résultats à l'avis de ses utilisateurs.

Chapitre 7

Étude comparée des algorithmes

Résumé

Dans ce chapitre, nous tentons d'évaluer la meilleure stratégie d'exploration de l'espace de recherche pour notre problème. Après un bref aperçu des méthodes classiques d'évaluation utilisées dans le domaine de la RI, nous concluons que ce type de mesure n'est pas adapté à notre problématique. Nous comparons ensuite les trois méta-heuristiques définies dans les chapitres précédents en les mettant en concurrence avec un méta-moteur compilant les résultats de moteurs de recherche classiques. Enfin, nous faisons évaluer le meilleur des algorithmes de recherche que nous avons conçus par des experts afin de déterminer objectivement son efficacité.

7.1 Évaluations classiques en RI

Le but de la RI est de trouver des documents pertinents à une requête, et donc utiles pour l'utilisateur. La qualité d'un système doit être mesurée en comparant les réponses du système avec les réponses idéales que l'utilisateur espère recevoir. Plus les réponses du système correspondent à celles que l'utilisateur espère, meilleur est le système.

Au fur et à mesure de l'évolution de ce domaine de recherche, des méthodes standard de mesure de qualité ont été mises au point afin de pouvoir comparer aisément les divers algorithmes de RI. La mesure de précision et de rappel est très utilisée sur des corpus textuels lorsque l'on connaît l'ensemble des éléments du corpus analysé. Cependant, cette méthode est difficilement applicable dans le cas d'un moteur de recherche car il est difficile d'avoir une idée précise de l'ensemble des documents atteignables. Pour résoudre ce problème, les conférences TREC ont vu le jour. Elles rassemblent de grandes collections de documents et de requêtes dont les réponses sont connues. Elles permettent ainsi d'évaluer objectivement l'efficacité des algorithmes de recherche. Cependant, ces

corpus possèdent quelques limitations qui nous empêchent de les utiliser dans notre méthodologie de test.

Dans la suite de ce chapitre, nous allons décrire ces deux éléments en identifiant les problèmes qu'ils nous posent dans l'évaluation de notre méthode.

7.1.1 Précision - Rappel

Les mesures d'évaluation basées sur la notion de *précision* et de *rappel* comptent parmi les plus anciennes du domaine de la RI. La précision est le rapport du nombre de documents pertinents correctement identifiés pertinents (vrais positifs TP) sur le nombre total de documents identifiés comme pertinents (TP + faux positifs FP) ; le rappel est le rapport du nombre de vrai positifs sur le nombre total de documents pertinents (TP + faux négatifs FN). À ces deux notions se rajoute une mesure d'efficacité globale, la *F-measure* initiée par Van Risbergen [Van Rijsbergen, 1979], qui représente la moyenne harmonique pondérée entre précision et rappel.

$$p = \frac{TP}{TP + FP},\ r = \frac{TP}{TP + FN}\ et\ F_\beta = \frac{\left(1 + \beta^2\right) \cdot p \cdot r}{\beta^2 p + r} \tag{7.1}$$

Des notions complémentaires peuvent apparaître dans la littérature : le $silence = 1 - rappel$ et le $bruit = 1 - précision$.

Idéalement, on voudrait qu'un système donne de bons taux de précision et de rappel en même temps. Un système qui obtenant 100% de précision et de rappel trouve tous les documents pertinents sans n'en citer aucun hors sujet.

Il existe une forte relation entre ces deux métriques comme le montre l'équation 7.1. Lorsque la précision augmente, le rappel diminue et inversement. Par conséquent, l'efficacité d'un système ne peut être déterminée que par l'utilisation d'une seule de ces mesures. Il est en effet facile d'obtenir 100% de rappel en donnant toute la base de test en réponse, mais dans ce cas, la précision serait quasi nulle. De même, si très peu de documents sont retournés en réponse, la précision pourrait être importante mais le rappel très faible.

Un système de recherche peut fournir une liste de documents résultats de taille variable. Par conséquent, la qualité d'un système de recherche est établie en effectuant plusieurs mesures de rappel en faisant varier le nombre de documents pris en compte.

Cependant, dans le cas d'un moteur de recherche, les documents pris en compte sont Internet dans sa globalité. Il est par conséquent impossible de connaître l'ensemble du corpus et de l'analyser pour déterminer les documents correspondant à une requête donnée. Ainsi on ne peut connaître ni l'ensemble des TP ni l'ensemble des FP.

Pour pallier à ce défaut, des corpus de tailles non négligeables ont été construits et analysés en fonction de requêtes précises. Cela permet d'établir sur des corpus de quelques dizaines voire quelques centaines de milliers de documents ceux qui sont pertinents à une requête et ceux qui ne le sont pas. Les conférences *TREC* ont vu le jour

dans le but de proposer à la communauté scientifique des bases nécessaires à l'évaluation des outils de recherche.

7.1.2 Les conférences *TREC*

Ces conférences ont été initiées par Harman en 1992 et ont pour objectif de tester des méthodes et des systèmes de RI avec des collections de textes de grandes tailles [Harman, 1993]. Chaque année les tâches à accomplir changent afin de s'adapter à la demande des différents organismes de recherche. De grandes bases de textes sont proposées accompagnées de requêtes prenant la forme du thème de la conférence - par exemple requêtes à base de questions ou mots-clés.

Les domaines abordés par ces conférences sont multiples et concernent notamment la RI ad hoc (la tâche classique de RI - soumettre des requêtes sur une collection statique), le filtrage de l'information, la RI non anglais (en espagnol, français, chinois) et translinguistique (retrouver des documents dans une langue différente de celle de la requête), la question-réponse, la RI multimédia (vidéo et parole).

L'évaluation d'un système particulier se fait en comparaison avec tous les autres outils de recherche participant à ces conférences. Par conséquent, elle permet de distinguer les meilleures algorithmes de recherche à un moment particulier.

Cependant, les requêtes proposées dans cette conférence sont figées et sont adaptées à ce que l'on retrouve classiquement dans les moteurs de recherche. Notre approche est différente en terme de requête. On désire que l'utilisateur spécifie au maximum ses souhaits grâce à diverses options comme présenté dans la section 3.3.2 du chapitre 3. Notre système est entièrement basé sur l'utilisation d'une requête riche afin de mieux analyser les documents rencontrés, ce qui n'est pas actuellement faisable dans un moteur de recherche classique. Donc nous ne pouvons évidemment pas utiliser ces corpus dans l'évaluation de notre méthode.

7.2 Étude comparative de trois méta-heuristiques

Afin de déterminer l'efficacité relative des différentes méta-heuristiques présentées dans les chapitres 5 et 6, nous avons utilisé la même méthodologie de tests qui nous a permis d'étudier les paramètres de ces trois algorithmes (voir section 5.3.2 du chapitre 5).

Pour toutes les requêtes présentées dans le tableau 5.1 page 138, nous avons exécuté 10 fois chacun des algorithmes. En complément, nous avons réalisé un méta-moteur de recherche simple qui compile les résultats des moteurs de recherche utilisés par l'opérateur O_{rand} (voir section 3.2 du chapitre 3) en les triant selon la fonction d'évaluation. Ce méta-moteur nous permet de juger de la qualité des résultats obtenus par les moteurs de recherche classiques. Nous avons pris en compte le même critère d'arrêt quelle que soit la méthode utilisée : la recherche prend fin lorsque 1000 pages ont été téléchargées.

Nous avons réuni les résultats obtenus dans les tableaux 7.1, 7.2 et 7.3 correspondant respectivement à l'analyse des 30, 60 et 100 premiers résultats obtenus par chaque

méthode. Pour chaque méthode, nous donnons la moyenne ainsi que l'écart type obtenu sur 10 essais. L'algorithme mis en jeu dans le méta-moteur est déterministe. Une seule exécution est par conséquent nécessaire pour chaque requête.

$\mathbf{n}^o$	Méta-recherche	GeniminerII		Antsearch		Tabusearch	
		moy	σ	moy	σ	moy	σ
1	81.37	69.99	[9.76]	▶ 89.60	[7.41]	87.43	[6.31]
2	88.36	▶ 88.99	[7.63]	38.87	[4.38]	51.19	[3.79]
3	74.29	▶ 92.80	[4.06]	54.28	[3.63]	55.13	[2.76]
4	▶ 100.00	80.04	[9.41]	79.08	[6.12]	62.86	[12.03]
5	85.64	▶ 87.93	[8.17]	46.06	[3.99]	38.05	[11.40]
6	▶ 100.00	88.45	[7.11]	52.23	[8.38]	62.65	[4.52]
7	▶ 99.27	97.71	[1.69]	66.67	[4.22]	71.95	[2.93]
8	▶ 100.00	72.62	[6.15]	30.72	[5.28]	50.95	[2.95]
9	67.56	▶ 91.71	[4.70]	55.18	[5.47]	50.86	[4.99]
10	▶ 100.00	93.37	[4.87]	58.20	[3.36]	58.39	[4.72]

TAB. 7.1 – Évaluation comparée sur 10 requêtes des 30 premiers résultats obtenus par *GeniminerII*, *Antsearch*, *Tabusearch* et par un méta-moteur compilant les résultats obtenus par les moteurs de recherche. Un "▶" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

Les résultats obtenus en analysant les 30 premiers documents (voir table 7.1) indiquent clairement que les méthodes à base de fourmis artificielles et d'algorithme tabou que nous avons réalisées ne parviennent pas à rivaliser avec l'approche génétique ou les résultats issus des moteurs de recherche classiques. On note toutefois une exception concernant la première requête pour laquelle la recherche à base de fourmis obtient les meilleurs résultats suivis de près par la méthode tabou.

Ces deux méthodes effectuent une exploration locale plus profonde que l'algorithme génétique. Or, le sujet de cette requête correspond à beaucoup de pages sur Internet et chacune de ces pages possède un grand nombre de liens vers d'autres documents portant sur le même sujet. Les documents considérés intéressants par notre fonction d'évaluation sont classés assez loin dans les moteurs de recherche mais des liens partant des résultats de ces moteurs permettent d'y accéder plus rapidement. L'algorithme génétique obtient ici les moins bons résultats car il n'a pu suivre assez longtemps le chemin menant aux meilleurs documents et n'a pas eu le temps de consulter les résultats éloignés dans les moteurs de recherche classiques.

D'une manière générale, *Antsearch* et *Tabusearch* obtiennent les mêmes résultats, chacun dominant l'autre sur 4 requêtes et pouvant être considérés sensiblement similaires sur les requêtes 3 et 10 (moins de .9 points d'écart).

Pour les autres requêtes, les meilleurs résultats sont partagés entre le méta-moteur et *GeniminerII* obtenant respectivement la tête du classement pour 5 et 4 requêtes. Les écarts vont d'une quasi égalité pour la requête 2 à des écarts bien plus grands avec plus de 27 points pour la requête 8. Cette dernière requête n'est pas le point fort des trois

méta-heuristiques que nous avons conçues qui réalisent des scores très faibles. Les termes contenus dans cette requête sont en effet très précis (vocabulaire de la programmation informatique) et il est par conséquent facile pour un indexeur d'obtenir des documents traitant de ce sujet. La constatation est la même pour la requête 4 issue d'un vocabulaire similaire pour laquelle les écarts entre le méta-moteur et les trois méta-heuristiques sont également importants. La situation est inversée pour les requêtes 3 et 9 où l'écart est à l'avantage de *GeniminerII*. Ces requêtes comportent des termes qui peuvent se retrouver dans beaucoup de documents (notamment de vente en ligne pour la requête 3) et l'analyse plus profonde des pages effectuée par notre méthode permet de détecter les documents plus intéressants vis-à-vis de la requête proposée.

$\mathbf{n}^o$	Méta-recherche	GeniminerII		Antsearch		Tabusearch	
		moy	σ	moy	σ	moy	σ
1	71.65	72.88	[6.35]	▶ 93.49	[6.30]	87.39	[2.29]
2	91.44	▶ 92.98	[6.84]	45.76	[4.92]	58.59	[4.09]
3	71.25	▶ 95.46	[2.85]	74.13	[3.26]	66.23	[4.53]
4	▶ 100.00	83.42	[8.28]	82.68	[2.04]	74.14	[6.09]
5	89.70	▶ 91.76	[4.09]	49.66	[6.73]	49.76	[16.08]
6	▶ 100.00	90.12	[4.09]	53.68	[10.42]	64.67	[2.93]
7	▶ 98.08	95.66	[3.10]	67.50	[3.54]	72.31	[1.79]
8	▶ 100.00	59.26	[7.09]	42.87	[7.65]	n.a.	
9	81.37	▶ 91.22	[4.79]	55.39	[4.57]	65.13	[5.63]
10	▶ 100.00	93.48	[3.09]	51.39	[4.89]	58.84	[6.11]

TAB. 7.2 – Évaluation comparée sur 10 requêtes des 60 premiers résultats obtenus par *GeniminerII*, *Antsearch*, *Tabusearch* et par un méta-moteur compilant les résultats obtenus par les moteurs de recherche. Un "▶" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

Le tableau 7.2 présente le même type de résultats mais prend en considération un nombre plus important de résultats : les 60 premiers documents. On retrouve ici les mêmes tendances que sur l'analyse effectuée sur les 30 premiers résultats.

La méthode tabou n'a pas été en mesure de produire 60 documents résultats sur la requête 8. Cette requête possède en effet un critère spécifiant que tous les mots-clés doivent être présents dans les documents résultats. Certaines pages Web parcourues par la méta-heuristique ne remplissant pas cette condition, elles ne peuvent figurer parmi les réponses.

Globalement, les résultats obtenus par *Antsearch* et *Tabusearch* se sont sensiblement améliorés. Les écarts restent toutefois similaires avec ceux observés dans le tableau 7.1. On note cependant quelques nuances pour la requête 1 où *GeniminerII* a rejoint et même dépassé le score obtenu par la méta-recherche ainsi que pour la requête 8 où l'écart entre ces deux méthodes s'est agrandi et pour la requête 9 où au contraire l'écart s'est réduit de moitié.

$\mathbf{n}^o$	Méta-recherche	GeniminerII		Antsearch		Tabusearch	
		moy	σ	moy	σ	moy	σ
1	73.75	73.35	[4.24]	► 92.35	[5.16]	85.11	[2.46]
2	► 93.27	93.06	[6.75]	52.24	[3.28]	68.19	[3.89]
3	80.38	► 93.18	[2.19]	92.90	[4.29]	78.38	[5.37]
4	► 99.41	87.90	[6.45]	n.a.		n.a.	
5	86.98	► 87.38	[6.91]	50.41	[7.64]	58.81	[10.17]
6	► 100.00	88.35	[1.98]	56.40	[9.28]	63.77	[1.84]
7	► 100.00	91.45	[3.27]	66.19	[4.66]	71.57	[2.37]
8	► 100.00	48.19	[6.59]	n.a.		n.a.	
9	89.93	► 95.32	[3.70]	58.18	[2.76]	71.51	[3.69]
10	► 100.00	91.28	[2.53]	47.70	[5.35]	61.72	[3.56]

TAB. 7.3 – Évaluation comparée sur 10 requêtes des 100 premiers résultats obtenus par *GeniminerII*, *Antsearch*, *Tabusearch* et par un méta-moteur compilant les résultats obtenus par les moteurs de recherche. Un "►" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

Enfin le tableau 7.3 présente les résultats obtenus en analysant les 100 premiers documents fournis par chaque méthode. Les méta-heuristiques *Antsearch* et *Tabusearch* ont été en incapacité de fournir 100 résultats pour les requêtes 4 et 8.

La principale différence observée par rapport aux deux précédents tableaux se situe sur la requête 2. En effet, la méta-recherche prend la tête du classement avec une différence somme toute très légère avec *GeniminerII*. Les écarts que nous avions observés dans les deux tableaux précédents étaient du même ordre de grandeur mais avec des classements inversés. On constate également une nouvelle inversion entre *GeniminerII* et la méta-recherche pour la requête 1 avec des écarts toujours aussi faibles.

L'ensemble des autres résultats est similaire aux observations que nous avons faites sur les deux précédents tableaux. Une exception peut cependant être faite pour la méthode *Antsearch* qui améliore grandement ses résultats pour la requête 3 en se rapprochant du score obtenu par *GeniminerII*.

En conclusion, des trois méta-heuristiques que nous avons imaginées, seule *GeniminerII* est capable de rivaliser avec les résultats obtenus par le méta-moteur. On a pu constater que la méthode génétique était complémentaire aux moteurs de recherche classiques. Suivant le type de requête proposée par l'utilisateur, la recherche peut se révéler plus ou moins difficile pour chaque méthode.

Nous avons défini la fonction d'évaluation afin d'être le plus proche possible de ce que désire un utilisateur. Les tests que nous avons effectués se basent sur cette fonction d'évaluation. Mais dans notre problème, seuls des experts humains peuvent réellement décider de la supériorité d'une méthode sur l'autre. Nous présentons par conséquent dans la section suivante une comparaison entre la meilleure heuristique que nous avons élaborée (*GeniminerII*) et les moteurs de recherche utilisés par l'opérateur O_{rand} (voir

section 3.2 du chapitre 3) soumise au jugement des utilisateurs de notre système.

7.3 Comparaison entre notre modèle et les moteurs de recherche classiques par des experts

Nous cherchons à déterminer l'apport de ce système en comparaison avec les outils de recherche existants et la facilité d'utilisation et de compréhension des multiples options de la requête d'interrogation. Nous avons ainsi mis en place un protocole de test destiné à des utilisateurs réels permettant de déterminer la pertinence des résultats retournés par le système. La requête est spécifiée grâce à l'interface graphique présentée dans la figure 7.1 et reprend les éléments définis dans la section 3.3.2 du chapitre 3. À chaque exécution, deux recherches sont effectuées : une à l'aide de *GeniminerII* et une consistant à interroger les moteurs de recherche classiques utilisés par l'opérateur O_{rand} (voir section 3.2 du chapitre 3) et à afficher alternativement les résultats obtenus par chaque moteur dans leur ordre d'apparition.

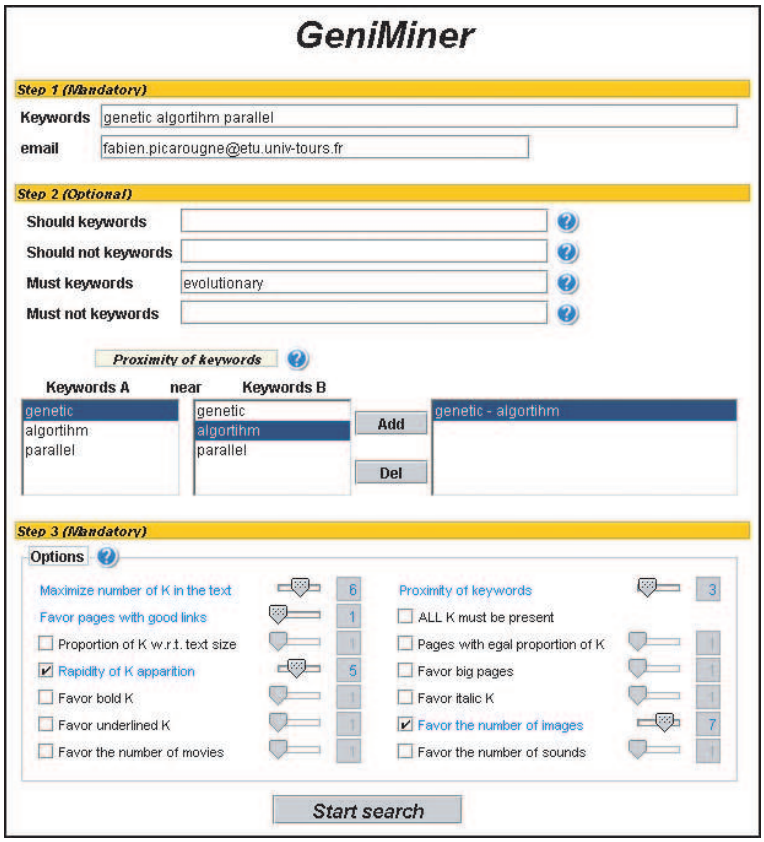


FIG. 7.1 – Interface graphique d'interrogation de *GeniminerII* sous forme d'applet Java.

Une fois les 100 premiers résultats obtenus, deux listes de liens et de résumés de pages Web (disposés côte à côte et correspondant aux deux exécutions de recherche

réalisées) sont retournées à l'utilisateur. Ce dernier a alors la possibilité d'attribuer une note de 1 à 10 à chaque liste indiquant le degré de pertinence des réponses à la requête initiale. La position des deux listes est définie aléatoirement de manière à ne pas biaiser les évaluations.

Nous avons rassemblé les résultats obtenus dans les tableaux 7.4 et 7.5. Pour chaque tableau, nous avons dissocié les requêtes utilisant uniquement les options par défaut (spécification uniquement des critères C_{K1}, C_S, C_{Sn}, C_M et C_{Mn}) des requêtes plus complexes permises par notre fonction d'évaluation. Ainsi, la deuxième ligne des tableaux ne prend en compte que ce dernier type de requête. Le tableau 7.4 présente le nombre de requêtes pour lesquelles une méthode de recherche particulière obtient un meilleur score. Alors que le tableau 7.5 indique le score cumulé obtenu par chaque méthode de recherche pour toutes les requêtes prises en compte.

Vote majoritaire	*GeniminerII*	*Méta-moteur*	*Égalité*
Toutes les requêtes	22 (44.00%)	20 (40.00%)	8 (16.00%)
Requêtes filtrées	13 (48.15%)	9 (33.33%)	5 (18.52%)

TAB. 7.4 – Évaluation par des utilisateurs réels de la pertinence des résultats retournés par *GeniminerII* en comparaison à un méta-moteur compilant les résultats issus de moteurs de recherche standards.

Vote majoritaire	*GeniminerII*	*Méta-moteur*
Toutes les requêtes	284 (49.65%)	288 (50.35%)
Requêtes filtrées	164 (50.62%)	160 (49.38%)

TAB. 7.5 – Évaluation par des utilisateurs réels de la qualité des résultats retournés par *GeniminerII* en comparaison à des moteurs de recherche standards.

GeniMinerII obtient une évaluation similaire voire légèrement supérieure (4 points) à celle obtenue par les moteurs de recherche classiques. Cet équilibre s'est vérifié des le début des évaluations et s'est confirmé tout au long de l'étude comme on peut le remarquer sur la figure 7.2. Dans ce graphe, nous observons l'évolution des votes dans un ordre chronologique et constatons une progression linéaire quasi identique de ceux accordés à *GeniminerII* et au méta-moteur. Ce résultat peut être rapproché avec ceux établis dans la section précédente. De plus, comme on peut le remarquer dans le tableau 7.5, la qualité moyenne des évaluations pour les deux requêtes est comparable. Notre méthode peut donc apporter des améliorations à la recherche produite par les outils classiques de recherche.

On peut également remarquer dans la deuxième ligne du tableau 7.4 que si on ne prend en compte que les requêtes exploitant un minimum les possibilités offertes par notre requête, *GeniminerII* obtient de meilleurs résultats (de l'ordre de 15 points). C'est la conclusion que nous souhaitions obtenir par une analyse plus précise des documents grâce à l'utilisation d'une requête plus riche.

Sur les requêtes pour lesquelles les utilisateurs ont considéré les méthodes équiva-

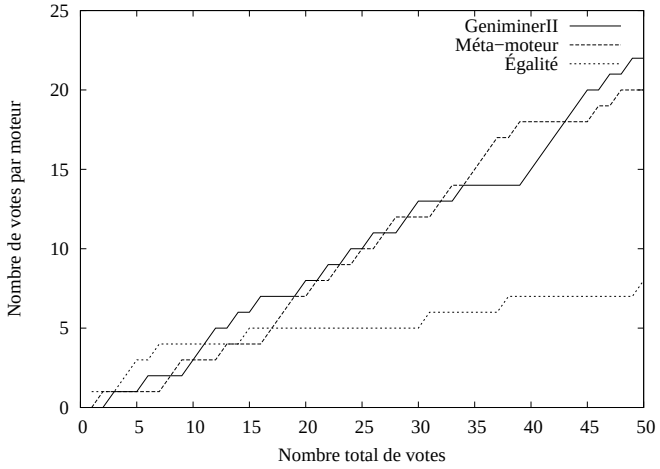


FIG. 7.2 – Évolution de l'évaluation de *GeniminerII* et d'un méta-moteur par des utilisateurs en fonction du nombre de votes effectués.

lentes (colonne Égalité du tableau 7.4), l'évaluation obtenue est majoritairement assez faible (pour 6 requêtes sur 8, elle est inférieure à 5 sur 10). Cela indique que la recherche demandée était mal formulée ou que les réponses pertinentes sont très difficiles à obtenir ou inexistantes. De plus, on a pu constater que les utilisateurs ont généralement quelques difficultés à assimiler le concept de pondération de critères de recherche. Près de la moitié des recherches ne prennent pas en compte les possibilités de spécification avancée offerte par la requête. Il faut signaler toutefois qu'il peut être difficile d'ajuster convenablement les poids des critères en fonction de la recherche effectuée. Cela demande une bonne connaissance du domaine de la recherche afin, par exemple, de déterminer efficacement les mots à associer dans une requête ou le poids à attribuer à la fréquence d'apparition des mots-clés dans le texte.

Parmi les votes ne donnant pas l'avantage à *GeniminerII*, certains utilisateurs nous ont indiqué que notre méthode fournissait parfois trop de résultats consécutifs appartenant au même domaine. Cela ne facilite pas la lecture des résultats, même si les documents correspondent à la demande initiale formulée par l'utilisateur. Ces remarques nous ont conduit à élaborer un système de classification des résultats que nous présentons dans le chapitre suivant.

Le système s'adresse ainsi clairement à des spécialistes, comme c'est le cas dans le domaine de la veille stratégique. Mais des améliorations peuvent être effectuées notamment en donnant une explication plus claire du fonctionnement de l'évaluation dans l'interface d'interrogation.

7.4 Conclusion

Dans ce chapitre, nous avons comparé plusieurs méta-heuristiques (une approche génétique, une approche à base de fourmis artificielles et une méthode tabou) adaptées au problème de la recherche d'information sur Internet. Les méthodes habituellement utilisées dans le domaine de la RI ne peuvent s'appliquer à notre modèle. Nous avons par conséquent établi un protocole de test permettant d'évaluer le plus objectivement possible ces trois approches et de les comparer aux résultats obtenus par les outils de recherche traditionnels.

Les résultats obtenus nous permettent de conclure à la supériorité de l'approche génétique sur les deux autres méta-heuristiques pour ce problème. De plus, pour l'ensemble des requêtes nous ayant servi à tester nos algorithmes, *GeniminerII* obtient des résultats comparables à un méta-moteur de recherche utilisant notre fonction d'évaluation pour trier ses résultats.

Nous avons ensuite voulu comparer les résultats obtenus par notre meilleure méthode aux résultats obtenus par les moteurs de recherche classiques. Pour cela nous avons demandé à plusieurs utilisateurs d'évaluer les résultats obtenus par chacune des méthodes pour une même requête. Une fois de plus aucune méthode ne prend le dessus sur l'autre ce qui montre la complémentarité des deux approches. Cette évaluation a également montré que lorsque l'utilisateur spécifie avec beaucoup de détails sa requête, notre modèle lui apporte globalement de meilleures réponses. Notre système permet en effet de spécifier une requête à l'aide de multiples critères utilisés dans l'analyse des documents rencontrés lors d'une recherche. Un moteur de recherche classique ne peut analyser les pages Web avec autant de précision du fait des contraintes qui lui sont imposées : nombre très important de requêtes simultanées et rapidité de réponse exigée. Notre outil de recherche se révèle par conséquent complémentaire aux moteurs de recherche classiques.

Lors des évaluations réalisées avec des utilisateurs réels de notre système, certaines remarques ont été émises notamment sur le besoin de classification des résultats. Nous présentons par conséquent dans le chapitre suivant une méthode permettant de regrouper de manière visuelle des résultats similaires obtenus par notre moteur de recherche.

Chapitre 8

Vers la visualisation de résultats par nuages d'agents

Résumé

Dans ce chapitre, nous présentons une classification originale des résultats d'un moteur de recherche. Cette classification est inspirée du comportement observé chez certains animaux se déplaçant en groupe d'individus. Chaque individu réagit à des règles en fonction de son voisinage. La classification consiste à regrouper les individus ayant des points communs et à les éloigner des autres. Les documents résultats sont représentés par groupes similaires et triés en fonction de la pertinence à la requête de l'utilisateur.

Comme nous l'avons vu dans le chapitre 1, la manière de présenter les résultats influence beaucoup leur compréhension par l'utilisateur. Dans la section 1.1.5 nous avons constaté que lorsque l'on présente les documents les plus pertinents sous forme d'une liste de liens accompagnée d'un résumé, les utilisateurs ne portent majoritairement leur attention que sur deux ou trois documents. Nous avons également vu dans la section 1.4 qu'une présentation différente, plus évoluée, permettait de comprendre davantage les résultats retournés par le moteur et de capter de manière significative l'attention de l'utilisateur. Un des moyens les plus efficaces consiste à regrouper les documents similaires. Cette méthode a la capacité de réduire l'effort d'analyse requis pour exploiter pleinement les résultats d'un moteur de recherche.

La classification est un des plus vieux problèmes traités dans le domaine informatique et bon nombre de méthodes d'origines diverses ont été élaborées. Nous avons observé dans la section 1.4.2.2 que certaines d'entre elles sont utilisées dans des moteurs de recherche. Comme nous l'avons indiqué dans le paragraphe précédent, le regroupement de résultats similaires apporte une information supplémentaire à l'utilisateur afin qu'il puisse interpréter au mieux les réponses d'un outil de recherche. Cependant, la présentation sous forme de catégories se fait classiquement par des listes en mode textes

(voir le modèle *Grouper* dans la section 1.4.2.2 du chapitre 1). Nous allons nous intéresser plus spécifiquement à des méthodes de classification visuelles. Nous avons par conséquent décidé d'utiliser une méthode de classification s'inspirant du déplacement des nuages d'insectes ou d'oiseaux évoquée dans la section 1.4.3.5. Notre choix s'est porté naturellement sur cette méthode parce qu'elle effectue une classification visuelle et dynamique en montrant l'évolution des regroupements à l'utilisateur. Elle fournit de plus des résultats très rapidement ce qui nous permet d'envisager une exécution sur le navigateur de l'utilisateur du système.

Dans la suite de ce chapitre, nous examinerons en premier lieu un algorithme de classification capable de regrouper des données indépendamment de leur type et s'inspirant du comportement de navigation observé dans les nuages d'agents. Nous verrons ensuite dans la section 8.2 comment adapter les résultats fournis par le moteur de recherche afin de pouvoir appliquer l'algorithme de classification. Enfin, nous présenterons des exemples de résultats obtenus par cette méthode dans la section 8.3.

8.1 Classification par nuages d'agents

De nombreux algorithmes de classification utilisent des principes inspirés de la biologie. Plusieurs méthodes de classification des résultats interprétables visuellement ont été testées (voir un survol de différentes méthodes dans [Azzag *et al.*, 2004]). Citons par exemple les cartes de Kohonen [Kohonen, 1989] mais dont l'apprentissage prend un certain temps. Les algorithmes génétiques ont été appliqués à la classification en utilisant différents codages [Cucchiara, 1993, von Laszewski, 1991, Jones et Beltrano, 1991, Falkenauer, 1994]. Un individu code alors le regroupement des données en classes, et cet individu évolue selon les principes de sélection, de recombinaison et de mutation modélisés à partir de la théorie du Darwinisme. De même, les algorithmes à base de population de fourmis artificielles ont modélisé la manière dont ces insectes sont capables d'organiser des objets en groupes selon leur similarité [Lumer et Faieta, 1994, Kuntz *et al.*, 1997, Monmarché *et al.*, 1999b] mais la proximité des groupes obtenus porte à confusion. Ils résolvent cependant un problème de partitionnement en utilisant des heuristiques particulières qui diffèrent notamment des opérateurs de croisement et de mutation utilisés dans les approches évolutionnaires.

Ces algorithmes biomimétiques ont pour principal intérêt d'effectuer un regroupement de manière distribuée et donc sans contrôle central. Ils ne nécessitent généralement pas de classification initiale des données ni même d'un nombre de classes connu a priori, comme c'est le cas par exemple pour l'algorithme des centres mobiles (k-means) détaillé dans [Jain et Dubes, 1988]. Ils peuvent généralement traiter aussi bien des données numériques que symboliques, et avoir un fonctionnement incrémental. Il existe encore d'autres algorithmes inspirés de systèmes biologiques mais qui n'ont pas encore été appliqués et testés sur des problèmes de classification. On peut citer par exemple les automates cellulaires ou encore les nuages d'animaux volants ou nageants.

Nous nous concentrons ici sur ce dernier modèle qui s'inspire du comportement observé chez certains animaux se déplaçant en groupes d'individus. Chaque individu réagit à des règles, souvent identiques à toute la population, qui ne prennent en compte que le voisinage immédiat de l'individu. En effet, ce dernier ne peut pas percevoir l'ensemble du nuage mais seulement les individus l'entourant. À partir du déplacement local de l'individu, des formes complexes pour le nuage d'animaux vont pouvoir apparaître, formes qui dépassent le cadre de la règle locale. Cette propriété d'émergence est recherchée dans de nombreux problèmes d'informatique puisqu'elle va notamment permettre à la fois une parallélisation décentralisée et massive de l'algorithme avec des comportements élémentaires simples tout en obtenant globalement un résultat complexe.

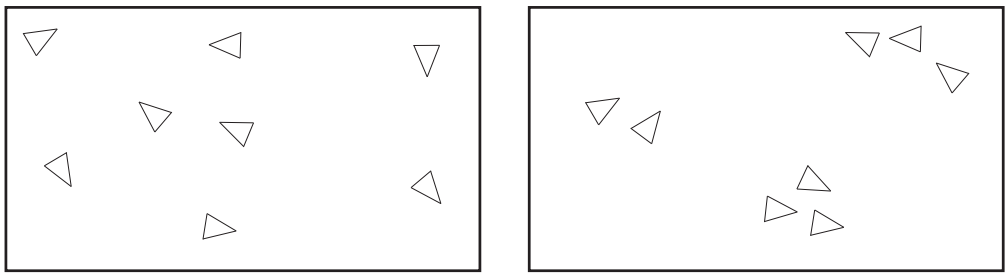
Dans le domaine de la classification, la manière dont les principes des nuages d'animaux volants vont être utilisés peut être décrite intuitivement de la façon suivante : chaque individu va représenter une donnée. Les individus vont être placés dans un environnement 2D ou 3D et sont caractérisés par trois éléments : leurs coordonnées dans l'environnement, leurs vecteurs vitesse et des règles comportementales pour gérer les déplacements. Ces règles sont communes à tous les individus et utilisent le voisinage local d'un agent pour décider de changer le vecteur vitesse. Elles vont prendre en compte la similarité des données portées par les agents afin de former des nuages de données homogènes. Autrement dit, la proximité des agents, à la fois en termes de position et de vitesse dans l'espace des déplacements 2D ou 3D, va correspondre à la similarité des données dans leur espace de description multidimensionnel. Non seulement ce type d'algorithmes va pouvoir classer efficacement des données sans connaissances initiales, mais en plus, permettre à l'expert du domaine d'obtenir une visualisation dynamique dans laquelle les distances entre données auront un sens du point de vue de la représentation des données (voir figure 8.1).

8.1.1 Le modèle biologique

Dans la nature, on observe par exemple que des nuages d'oiseaux peuvent se former et évoluer de manière très spectaculaire et utile pour les individus ayant ce comportement : l'effet de masse peut repousser des prédateurs éventuels ou alors certaines configurations précises permettent d'économiser de l'énergie (exemple du vol des canards). Des comportements similaires se retrouvent dans la nage collective de certaines espèces de poissons pour la chasse en groupe par exemple (banc de thons) où l'essaim prend la forme d'un entonnoir ou pour échapper à un ennemi avec des mouvements d'explosion, ou encore dans le vol en groupe chez des insectes (criquets). Enfin, le déplacement des poussins en train de picorer présente une forme simple de mouvement auto-organisé qui leur permet de rester à une certaine distance les uns des autres grâce aux signaux que constituent leurs pépiements. Dans la littérature sur la biologie, de nombreux articles relatent de ce type de comportements collectifs pour les déplacements [Aoki, 1984, Breder, 1951, Breder, 1959, Davis, 1975, Davis, 1980, Camazine *et al.*, 2001].

En informatique, les modélisations directement issues de ces comportements sont caractérisées par une «intelligence en essaim» («swarm intelligence») qui fait qu'un groupe d'entités plutôt simples et obéissant à des règles locales de coordination pour

leurs déplacements sont capables d'engendrer des comportements et des mouvements globaux beaucoup plus complexes [Bonabeau *et al.*, 1999].



a) *b)*

FIG. 8.1 – Illustration de l'environnement dans lequel va se déplacer le nuage d'agents et du passage progressif de la situation initiale *a)* où les agents sont placés aléatoirement et orientés dans des directions désordonnées à une organisation finale *b)* en groupes se déplaçant de manière cohérente.

Pour la classification, l'idée est de considérer que des individus représentent les données à regrouper et qu'ils se déplacent suivant une règle de comportement local telle que, après un certain nombre de déplacements, des groupes d'agents homogènes se forment en se déplaçant ensemble (voir figure 8.1).

8.1.2 Principes des déplacements

8.1.2.1 Algorithme principal

Nous considérons un ensemble de n données $(e_1, ...e_n)$ que l'on souhaite regrouper en classes. Nous ne faisons pas d'hypothèses particulières quant à l'espace de représentation de ces données qui peuvent être aussi bien numériques, symboliques ou encore textuelles. La seule condition que nous imposons est l'existence d'une fonction de similarité dont les valeurs sont comprises dans l'intervalle $[0, 1]$, la borne supérieure correspondant à deux données complètement identiques. Pour deux données e_i et e_j, on note cette fonction de similarité $\mathrm{Sim}(i, j)$. Nous détaillerons dans la section 8.2 la fonction de similarité utilisée pour traiter les résultats provenant du moteur de recherche.

Nous considérons une population de n agents où l'agent i représente la donnée e_i. Les agents se déplacent dans un espace de dimension 2 comme représenté sur la figure 8.1. Un agent i est caractérisé par ses coordonnées réelles $(x_i, y_i) \in [0, 1] \times [0, 1]$ ainsi que par un vecteur vitesse $\mathbf{v}_i = A_i\hat{\mathbf{v}}_i$ d'amplitude A_i ($A_i \geq 0$) et de direction normalisée $\hat{\mathbf{v}}_i$ ($||\hat{\mathbf{v}}_i|| = 1$). On note par $d(i, j)$ la distance euclidienne 2D entre deux agents i et j. L'algorithme principal (algorithme 8.1) fonctionne de la façon suivante : les agents sont placés initialement à une position choisie aléatoirement et avec une direction initiale $\hat{\mathbf{v}}_i$ aléatoire. Les distances idéales sont alors calculées pour chaque couple d'agents (voir la section 8.1.2.2). Ensuite, au gré de leur déplacement, les agents vont se croiser et vont décider suivant une règle de comportement local de se rapprocher

(1) Lire les n données d'entrée $e_1, ..., e_n$
(2) Placer initialement les n agents de façon aléatoire dans l'environnement 2D
(3) Calculer la distance idéale entre chaque couple d'agents
(4) **tantque** itération $<$ NbItération **faire**
(5) Calculer un déplacement pour chacun des n agents en fonction d'une règle locale
(6) Déplacer tous les agents simultanément
(7) **fintantque**
(8) Construire éventuellement des classes à partir des regroupements d'agents et Donner ces classes en sortie

ALG 8.1: Algorithme décrivant le principe général de la classification par nuage d'agents.

ou de s'éloigner les uns des autres, d'aller ou non dans les mêmes directions. Cette règle locale tend intuitivement vers deux buts : (1) établir une distance 2D entre agents qui soit représentative de la similarité des données qu'ils représentent, (2) déplacer dans la même direction des agents représentant des données similaires. À partir de cette règle locale vont émerger des groupes d'agents se déplaçant ensemble (voir figure 8.1 *b)*) et permettant de définir des regroupements dans les données.

8.1.2.2 Règle de comportement local

Le comportement d'un agent est déterminé par une règle individuelle, identique pour tous les agents et décrite par l'algorithme 8.2. Cette règle calcule pour un agent donné un nouveau vecteur vitesse en fonction de sa position et de sa vitesse actuelle et des autres agents situés dans son voisinage. Pour un agent i, il faut déterminer un voisinage $\mathcal{V}(i)$ correspondant à l'ensemble des autres agents j se trouvant à une distance euclidienne 2D $d(i,j)$ inférieure ou égale à d_{seuil} (ligne 2 de l'algorithme 8.2). Cette notion est prépondérante puisqu'elle définit le seuil en dessous duquel un agent peut en percevoir un autre et être influencé par celui-ci et réciproquement. Deux cas peuvent se produire : s'il n'y a aucun agent dans le voisinage de i, alors i va se déplacer dans la même direction $\hat{\mathbf{v}}_i$ que précédemment (ligne 5 de l'algorithme 8.2) avec l'amplitude A_i fixée à $\frac{1}{4}d_{\text{seuil}}$ (ligne 25). Si des agents sont présents dans le voisinage de i, alors l'influence de ces agents sur i peut être séparée en deux parties : d'une part le changement de direction du vecteur vitesse, et d'autre part le changement de l'amplitude de ce vecteur. Ces points sont détaillés dans les paragraphes suivants.

Changement de direction. Nous considérons dans un premier temps le cas où un seul agent j est présent dans le voisinage de i (lignes 7 à 19 de l'algorithme 8.2), puis nous généraliserons dans un deuxième temps au cas de plusieurs agents. L'influence de j sur i en terme de direction de déplacement dépend d'une part de la valeur de la distance $d(i,j)$ par rapport à une distance idéale entre les deux agents, notée $d^*(i,j)$, et d'autre

(1) Déterminer le voisinage de i :
(2) $\quad \mathcal{V}(i) = \{j / j \in [1,n], j \neq i, d(i,j) \leq d_{\text{seuil}}\}$
(3) Calculer une nouvelle direction $\hat{\mathbf{v}}_i(t+1)$:
(4) **si** $\mathcal{V}(i) = \emptyset$ **alors**
(5) $\quad \hat{\mathbf{v}}_i(t+1) \leftarrow \hat{\mathbf{v}}_i(t)$
(6) **sinon**
(7) $\quad$ **pour** chaque agent $j \in \mathcal{V}(i)$ **faire**
(8) $\quad\quad$ **si** $d(i,j) > d^*(i,j)$ **alors**
(9) $\quad\quad\quad \beta(i,j) \leftarrow 4 \times \left(\frac{d(i,j)-d^*(i,j)}{d_{\text{seuil}}-d^*(i,j)}\right)^2$ /* attraction */
(10) $\quad\quad$ **finsi**
(11) $\quad\quad$ **si** $d(i,j) = d^*(i,j)$ **alors**
(12) $\quad\quad\quad \beta(i,j) \leftarrow 0$
(13) $\quad\quad$ **finsi**
(14) $\quad\quad$ **si** $d(i,j) < d^*(i,j)$ **alors**
(15) $\quad\quad\quad \beta(i,j) \leftarrow -4 \times \left(1 - \frac{d(i,j)}{d^*(i,j)}\right)^2$ /* répulsion */
(16) $\quad\quad$ **finsi**
(17) $\quad\quad$ l'influence de j sur i est alors :
(18) $$\mathbf{v}_{\text{résultant}}(i,j) \leftarrow \underbrace{\hat{\mathbf{v}}_j}_{\text{alignement}} + \underbrace{\beta(i,j) \times \hat{\mathbf{v}}_{ij}}_{\text{attraction ou répulsion}}$$
(19) $\quad$ **finpour**
(20) $\quad$ Calculer la somme des influences :
(21) $$\mathbf{v}_{\text{résultant}}(i,\cdot) \leftarrow \sum_{j \in \mathcal{V}(i)} \mathbf{v}_{\text{résultant}}(i,j)$$
(22) $\quad$ Normaliser : $\hat{\mathbf{v}}_{\text{résultant}}(i,\cdot) \leftarrow \frac{\mathbf{v}_{\text{résultant}}(i,\cdot)}{||\mathbf{v}_{\text{résultant}}(i,\cdot)||}$
(23) $\quad$ Nouvelle direction : $\hat{\mathbf{v}}_i(t+1) \leftarrow \hat{\mathbf{v}}_{\text{résultant}}(i,\cdot)$ en limitant l'angle entre $\hat{\mathbf{v}}_i(t+1)$ et $\hat{\mathbf{v}}_i(t)$ à 90 degrés
(24) **finsi**
(25) Calculer une nouvelle amplitude : $A_i(t+1) = \frac{1}{5} \times d_{\text{seuil}} + \frac{1}{20}\frac{d_{\text{seuil}}}{|\mathcal{V}(i)|+1}$
(26) Calculer le déplacement de l'agent i (les agents sont déplacés tous en même temps) : $(x_i(t+1), y_i(t+1)) = (x_i(t), y_i(t)) + A_i(t+1)\hat{\mathbf{v}}_i(t+1)$

ALG 8.2: Algorihtme de calcul du déplacement d'un agent i

part de l'angle entre le vecteur direction de j et celui de i (notés $\hat{\mathbf{v}}_i$ et $\hat{\mathbf{v}}_j$).

L'idée de distance idéale développée ici est simple : la distance $d(i,j)$ doit converger vers $d^*(i,j)$ qui tient compte de la similarité entre e_i et e_j. Si cette similarité est importante et que les agents sont trop éloignés par rapport à la distance idéale, alors il faut les rapprocher. Inversement, si cette similarité est faible et que les agents sont trop proches, alors il faut les éloigner. Cette distance idéale dépend elle-même de la similarité $\mathrm{Sim}(i,j)$ entre les données e_i et e_j de la manière suivante :

$$d^*(i,j) = \frac{1 - \mathrm{Sim}(i,j)}{1 - \mathrm{Sim}_{\mathrm{seuil}}} \times d_{\mathrm{seuil}} \tag{8.1}$$

Autrement dit, si $\mathrm{Sim}(i,j)$ est exactement égale au seuil $\mathrm{Sim}_{\mathrm{seuil}}$, on va tenter de placer idéalement les deux agents exactement à la limite de leur voisinage ($d^*(i,j) = d_{\mathrm{seuil}}$). Si $\mathrm{Sim}(i,j) < \mathrm{Sim}_{\mathrm{seuil}}$ on va tenter de faire se séparer ces deux voisins ($d^*(i,j) > d_{\mathrm{seuil}}$). Au contraire, si $\mathrm{Sim}(i,j) > \mathrm{Sim}_{\mathrm{seuil}}$, on va laisser les deux agents dans leur voisinage et à une distance donnée ($d^*(i,j) < d_{\mathrm{seuil}}$). $\mathrm{Sim}_{\mathrm{seuil}}$ a été calculée par la formule $\mathrm{Sim}_{\mathrm{seuil}} = \frac{1}{2}(\mathrm{Sim}_{\mathrm{moyenne}} + \mathrm{Sim}_{\mathrm{max}})$ où $\mathrm{Sim}_{\mathrm{moyenne}}$ et $\mathrm{Sim}_{\mathrm{max}}$ correspondent respectivement à la similarité moyenne et maximum entre les données (ou une valeur approchée calculée sur 1000 couples de données choisies aléatoirement pour les grands ensembles de données).

Pour calculer l'influence de j sur la direction de i, il faut donc comparer la distance actuelle $d(i,j)$ entre les deux agents à la distance idéale $d^*(i,j)$, et prendre en compte les deux vecteurs vitesse de chacun des agents. Cette influence prend la forme d'un vecteur $\mathbf{v}_{\mathrm{résultant}}(i,j)$ qui, une fois normalisé, modifie la direction $\hat{\mathbf{v}}_i$ de l'agent i. Ce vecteur a deux composantes : l'une représentant un alignement entre les deux vecteurs vitesse et qui pousse i à aligner son déplacement sur j, l'autre représentant une attraction ou une répulsion entre les deux agents. On distingue donc les trois cas suivants : si la distance entre les deux agents est plus grande que la distance idéale ($d(i,j) > d^*(i,j)$), alors une attraction a lieu entre i et j (voir ligne 9) ; si $d(i,j) = d^*(i,j)$, alors le vecteur vitesse de i se rapproche simplement de celui de j en restant à la distance idéale (voir ligne 12) ; enfin, si $d(i,j) < d^*(i,j)$, alors une répulsion a lieu entre i et j puisque qu'ils sont trop proches l'un de l'autre (voir ligne 15).

L'influence de i sur j est alors calculée avec la formule générale (voir ligne 18) :

$$\mathbf{v}_{\mathrm{résultant}}(i,j) = \hat{\mathbf{v}}_j + \beta(i,j) \times \hat{\mathbf{v}}_{ij} \tag{8.2}$$

où $\hat{\mathbf{v}}_{ij}$ est un vecteur unitaire pointant de i vers j, et où $\beta(i,j)$ prend une valeur positive, nulle ou négative suivant les trois cas énumérés précédemment (voir la figure 8.2).

Ensuite, pour considérer le cas de plusieurs agents dans le voisinage de i, on effectue simplement la somme des vecteurs obtenus pour chacun des agents et on obtient le vecteur résultant $\mathbf{v}_{\mathrm{résultant}}(i,\cdot)$ qui est normalisé pour obtenir $\hat{\mathbf{v}}_{\mathrm{résultant}}(i,\cdot)$ (lignes 21 et 22 de l'algorithme de la figure 8.2). Ce dernier vecteur représente la nouvelle direction de déplacement souhaitée pour l'agent i. Il faut cependant considérer que les agents sont limités dans les changements de direction qu'ils peuvent réaliser (voir ligne 23).

L'effet de cette règle locale de comportement peut être constaté sur la figure 8.3 qui donne le résultat théoriquement prévisible et observé en pratique lorsque deux agents

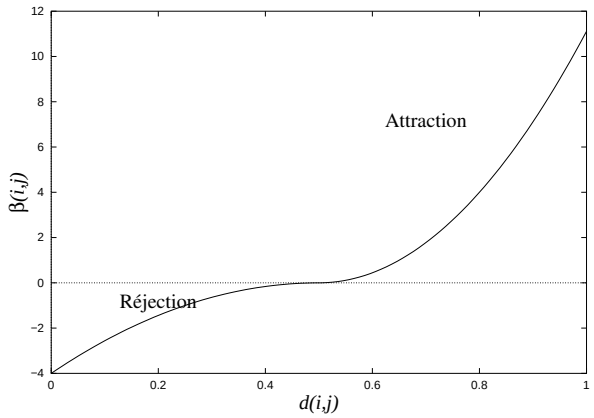


FIG. 8.2 – Allure de la courbe $\beta(i,j)$ calculée dans le cas d'une attraction ou d'une répulsion entre deux agents en fonction de la distance $d(i,j)$ (et en considérant pour clarifier cet exemple que la distance idéale entre i et j vaut la moitié de d_{seuil})

similaires vont se rencontrer (et en l'absence d'autres interactions). Les agents ont tendance à maintenir entre eux une distance égale à la distance idéale et à se déplacer dans la même direction.

Changement d'amplitude et déplacement final. Pour déterminer l'amplitude du vecteur vitesse d'un agent i, on applique la formule de la ligne 25 de l'algorithme 8.2. La première composante de l'amplitude fixe la vitesse minimum des agents à $\frac{1}{5}$ de la distance définissant le voisinage. Cela garantit d'abord qu'aucun agent ne restera immobile dans l'environnement sans faire de rencontre avec d'autres agents. De plus, un agent ne se déplacera pas suffisamment vite pour venir par exemple se placer directement au milieu d'un groupe. Cela pourrait en effet faire éclater un groupe homogène si l'intrus placé directement au centre est vraiment dissimilaire. Au lieu de cela, un agent sera progressivement rejeté ou attiré aux abords d'un groupe. La deuxième composante fait que des agents se déplaçant en un groupe homogène auront une vitesse légèrement plus faible que des agents seuls qui pourront ainsi naviguer plus vite et augmenter leurs chances de trouver un groupe qui les accepte.

Pour déplacer tous les agents en même temps, on considère qu'ils effectuent des mouvements toutes les unités temps (voir ligne 26). Les limites de l'environnement sont gérées de façon à ce que les agents tournent de manière symétrique par rapport à la bordure de l'espace ("rebondissement").

8.1.3 Optimisation de la complexité

8.1.3.1 Calcul du voisinage par matrice

Dans l'algorithme 8.2, le coût le plus élevé en terme de temps de calcul correspond au calcul du voisinage de chaque agent. Pour simuler les n déplacements, la construc-

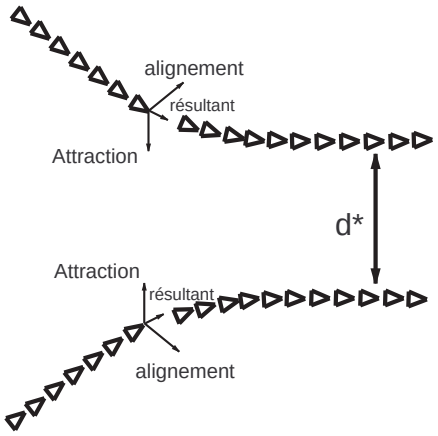


FIG. 8.3 – Comportement théorique de l'algorithme dans le cas de deux agents en interaction

tion de ces voisinages est en n^2 du fait qu'il faille, pour chaque agent, tester si tous les autres sont dans son voisinage ou non. De plus, les calculs permettant la mise à jour de la vitesse dépendent du nombre de voisins considérés. Nous avons mis en place les techniques suivantes qui permettent de ramener la complexité en n : avant de calculer les déplacements, les agents sont affectés aux cases de dimension $\frac{1}{d_{\text{seuil}}}$ d'une matrice carrée. Le calcul du voisinage d'un agent i consiste alors simplement à tester les 9 cases centrées autour de la case contenant i. Chaque case contient un nombre d'agents limité à 20 d'après nos expérimentations. Ce calcul est donc indépendant du nombre d'agents. Par contre, une approximation statistique est effectuée : si plus de 20 agents ont la même position dans la matrice (ce que l'on pourrait appeler en terme de hachage une collision), seulement 20 d'entre eux choisis aléatoirement seront considérés. Si peu d'agents sont présents dans le voisinage de i, cela ne change rien à l'exécution de l'algorithme car le risque de collision est faible. Si beaucoup d'agents sont présents, une partie d'entre eux ne sera pas considérée dans le calcul des voisinages et des interactions puisqu'il n'y aura pas assez de place dans la matrice. On peut argumenter néanmoins que cette approximation ne change pas globalement le comportement car lorsque beaucoup d'agents sont présents dans une zone réduite, un échantillonnage de cette population suffit statistiquement pour effectuer le calcul des interactions. D'un point de vue biologique, il a été envisagé que l'on trouvait ce même type de simplification dans les systèmes naturels [Camazine *et al.*, 2001]. Nous avons vérifié ce point expérimentalement sur deux bases de données réelles (*Soybean* et *Iris*) issues du Machine Learning Repository [Blake et Merz, 1998] et sur deux bases de données numériques artificielles (*Art1* et *Art2*) que nous avons générées.

Le temps de calcul est très largement amélioré, et ceci sans dégradation des performances de l'algorithme en terme de classification. Par exemple, la table 8.1 donne, pour un nombre d'itérations données (1000), pour quelques bases de données de taille croissante et sur une machine avec un processeur de type Intel Celeron à 650 Mhz, le

Données	#**Ex.** (n)	Temps sans optimisation	Temps avec optimisation
Soybean	47	3.40 (0.49)	2.80 (0.60)
Iris	150	18.00 (0.89)	7.30 (0.46)
Art1	400	173.10 (2.77)	29.00 (6.68)
Art2	1100	1304.90 (19.27)	113.80 (1.54)

TAB. 8.1 – Diminution du temps de calcul moyen mis pour effectuer 1000 itérations (sur 10 essais, écarts-types entre parenthèses, temps exprimés en secondes) obtenu grâce à l'optimisation du calcul du voisinage des agents.

temps nécessaire pour effectuer ces itérations. Pour une base de petite taille, l'amélioration est légère, et plus la taille de la base de données augmente, plus le gain obtenu est important.

8.1.3.2 Critère d'arrêt fondé sur l'entropie spatiale

Pour mieux quantifier la convergence de l'algorithme et savoir à quel moment il peut être arrêté sans espérer d'améliorations significatives, nous avons défini un critère qui mesure simplement l'état de désordre spatial du nuage d'agents. Ainsi, à partir des coordonnées continues des agents, nous définissons un quadrillage 20×20 de l'espace 2D ($[0,1] \times [0,1]$). La taille des cases a été choisie de façon à offrir un compromis entre un temps de calcul de l'entropie trop élevé et une précision suffisante. Pour chaque case (i,j) de ce quadrillage, on peut mesurer la proportion $p(i,j)$ d'agents présents. Le désordre du nuage à l'itération t est alors simplement défini par l'entropie spatiale inspirée de la définition de Shannon :

$$ES(t) = -\sum_{i=1}^{20}\sum_{j=1}^{20} p(i,j)\ \ln\ p(i,j) \tag{8.3}$$

Cette entropie va nous permettre de définir un critère d'arrêt pour l'algorithme qui va correspondre à un état le plus ordonné possible du nuage. Ce critère est donc le suivant : si le minimum de l'entropie n'a pas été amélioré depuis $3 * n$ itérations (n est le nombre d'exemples de la base), alors l'algorithme s'arrête. Nous avons fixé ce nombre d'itérations ainsi que la taille du quadrillage de manière expérimentale.

8.1.4 Algorithme de classification

Nous proposons un algorithme très simple pour calculer une classification effective des données en fonction des groupes d'agents qui se sont formés. Cet algorithme fonctionne de la façon suivante :

1. soit i un agent n'ayant pas encore de classe. On créé une nouvelle classe contenant seulement i ;

2. tous les agents du voisinage $\mathcal{V}(i)$, qui n'ont pas encore de classe définie et qui sont tels que $Sim(i,j) > Sim_{seuil}$, sont affectés à cette classe. On relance de manière récursive cette opération sur tous les agents ainsi ajoutés à la classe ;
3. lorsque plus aucun agent ne peut être affecté à cette classe, on recommence en 1 avec un nouvel agent sans classe. Si tous les agents ont été classés, l'algorithme passe à l'étape 4,
4. enfin, on traite notamment les cas d'agents restés isolés de la manière suivante : toute classe contenant moins de $\frac{n}{20}$ données est détruite, et les données ainsi libérées sont placées selon la donnée classée qui leur est la plus similaire.

8.1.5 Validation de l'algorithme

Afin de valider cet algorithme, nous l'avons évalué sur des données numériques. Nous avons utilisé pour cela 8 bases de données artificielles (Art1 à Art8) et quatre bases de données réelles issues du Machine Learning Repository [Blake et Merz, 1998]. Les données artificielles Art1, Art2, Art3, Art5 et Art6 sont engendrées par des lois gaussiennes avec des difficultés diverses (recouvrement des classes, attributs non pertinents, etc.), les données Art4 son engendrées par une loi uniforme, et enfin les données Art7 et Art8 correspondent à du bruit uniforme [Monmarché, 2000].

Nous utilisons le critère d'arrêt défini dans la section 8.1.3.2 pour obtenir un partitionnement des données. Comme nous avons choisi des données dont la classe est connue, il est possible de comparer la classification trouvée à la classification réelle des données. Cette comparaison a lieu en fonction du nombre de classes trouvées mais aussi en fonction d'une erreur de classification. Cette erreur consiste à énumérer tous les couples d'exemples de la base et à compter combien de couples sont mal classés. Un couple est considéré comme mal classé si les deux exemples sont classés ensemble alors qu'ils ne l'étaient pas dans la classification réelle, ou inversement. Cette erreur est divisée par le nombre total de couples $(n(n-1)/2)$.

Les résultats sont présentés dans le tableau 8.2. On peut remarquer que notre algorithme se comporte correctement sur les données testées : il retrouve un nombre de classes proche de la réalité, et regroupe dans ces classes la majorité des exemples correspondant aux classes d'origine.

8.2 Application à la classification des résultats d'un moteur de recherche

L'algorithme de classification ainsi défini permet de regrouper en catégories n'importe quel type d'éléments. Il suffit uniquement de définir une fonction de similarité des éléments pris en compte. Cette fonction établit le degré de similitude entre deux éléments en déterminant, pour chaque couple, une valeur numérique comprise dans l'intervalle $[0,1]$. Deux entités identiques obtiendront une similarité de 1.

Données	**#Ex.**	**#Attr.**	**#Cl. réelles**	**#Cl. trouvées**	**Erreur classif.**	**Nb itér.**	**Temps exéc. (s)**
Art1	400	2	4	4.5 (0.8)	0.19 (0.04)	2950.1 (744.3)	117.1 (31.9)
Art2	1000	2	2	3.6 (0.8)	0.19 (0.08)	8299.9 (3728.7)	986.8 (459.4)
Art3	1100	2	4	3.3 (1.0)	0.26 (0.13)	7315.9 (1709.6)	954.4 (255.9)
Art4	200	2	2	5.2 (1.1)	0.25 (0.06)	1952.3 (341.5)	31.7 (6.7)
Art5	900	2	9	5.6 (1.0)	0.21 (0.06)	5956.4 (1173.8)	563.8 (87.8)
Art6	400	8	4	4.2 (0.9)	0.05 (0.05)	3273.5 (653.9)	123.8 (23.9)
Art7	100	2	1	5.5 (1.0)	0.78 (0.06)	1002.7 (368.8)	5.2 (2.2)
Art8	1000	2	1	4.7 (1.2)	0.70 (0.09)	7338.8 (2290.6)	807.6 (255.5)
Soybean	47	35	4	5.6 (1.7)	0.12 (0.06)	601.9 (204.1)	1.3 (0.6)
Iris	150	4	3	4.1 (1.0)	0.21 (0.04)	1774.2 (474.5)	15.7 (4.4)
Pima	768	8	2	1.7 (0.8)	0.47 (0.02)	4445.4 (1276.4)	326.0 (91.8)
Thyroid	215	5	3	2.2 (1.1)	0.43 (0.07)	1982.0 (504.5)	40.9 (11.8)

TAB. 8.2 – Données utilisées et résultats quantitatifs obtenus avec un algorithme de classification utilisant les nuages d'agents (avec répartition initiale aléatoire des agents et $d_{\text{seuil}} = 0.04$). Les résultats sont donnés en moyenne sur 10 essais (écarts-types entre parenthèses)

Nous cherchons à classer des résultats de moteurs de recherche en catégories. Ces résultats se présentent sous forme de textes. Il existe plusieurs méthodes de mesure de similarité de textes, mais généralement, la mesure *cosine* est considérée comme celle apportant les meilleurs résultats. Cette mesure s'appuie sur une représentation vectorielle des différents documents à analyser. Nous avons détaillé cette représentation et cette méthode dans la section 1.1.3 du chapitre 1.

Dans le modèle vectoriel, chaque document est représenté par un vecteur de termes dont la dimension correspond au nombre de mots uniques compris dans le document. On fait correspondre à chaque terme, et donc à chaque composante du vecteur, un poids dépendant de la fréquence du terme dans le document. Plus précisément, nous déterminons ces poids par la méthode *tf*idf* décrite également dans la section 1.1.3. Cette méthode détermine le poids d'un terme en mesurant son importance dans le document (composante tf) et son pouvoir de discrimination par rapport aux autres

documents pris en compte (composante *idf*). La formule suivante détaille chacune de ces composantes.

$$tf = log\left(f(t_i, d) + 1\right) \tag{8.4}$$
$$idf = log(1/n) \tag{8.5}$$

$f(t_i, d)$ correspondant à la fréquence d'occurrence du terme t_i dans le document d et n représentant le nombre de documents du corpus dans lequel figure le terme t_i.

Préalablement au calcul du poids des termes du vecteur, nous éliminons les mots apportant le moins d'information. La loi de Zipf permet de distinguer statistiquement les mots les plus significatifs dans un ensemble de documents (voir section 1.1.3). En effet, les mots les plus fréquents et les moins fréquents ne sont généralement pas porteurs d'information permettant de distinguer le sens d'un document.

Une fois toutes les composantes des vecteurs de documents calculées, nous établissons la fonction de similarité en appliquant la mesure *cosine* donnée dans la formule 8.6 élevée au carré. Dans cette formule, $d_{i,n}$ désigne le terme n du document i.

$$\mathrm{Sim}(d_i, d_j) = cos^2(d_i, d_j) = \left(\frac{\sum_n d_{i,n} d_{j,n}}{\sqrt{\sum_n d_{i,n}^2 \sum_n d_{j,n}^2}} \right)^2 \tag{8.6}$$

Cette mesure est indépendante de la norme utilisée pour le calcul des poids des termes. En effet le cosinus ne produit que des valeurs comprises dans l'intervalle $[-1, 1]$. La mesure utilisée ($cosine^2$) respecte ainsi le critère requis par notre algorithme de classification : établir une valeur de similarité comprise dans l'intervalle $[0, 1]$.

Ainsi, nous pouvons construire la matrice de similarité regroupant l'ensemble des mesures de similarités calculée sur tous les documents deux à deux. Il s'agit par conséquent d'une matrice symétrique dans laquelle la diagonale n'est composée que de 1. L'exemple suivant illustre une telle matrice.

$$\begin{array}{c} \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \cdots & d_n \end{array} \\ \begin{pmatrix} 1 & \mathrm{Sim}(d_1, d_2) & & \mathrm{Sim}(d_1, d_n) \\ \mathrm{Sim}(d_1, d_2) & \ddots & & \mathrm{Sim}(d_2, d_n) \\ & & \ddots & \\ \mathrm{Sim}(d_1, d_n) & \mathrm{Sim}(d_2, d_n) & & 1 \end{pmatrix} \end{array} \tag{8.7}$$

À partir de cette matrice de similarité, l'exécution de l'algorithme de classification par nuage d'agents ne pose aucun problème particulier.

8.3 Exemples de résultats obtenus avec *GeniminerII*

Afin de tester l'apport de la classification par nuages d'agents sur les résultats issus de notre moteur de recherche, nous analysons les groupes formés à l'issue de trois requêtes. Les requêtes prises en compte ont été choisies parmi celles présentées dans le tableau 5.1 page 138.

Pour chaque requête testée, nous avons utilisé l'algorithme *GeniminerII* (décrit dans le chapitre 5) afin d'obtenir les documents correspondant le mieux au thème abordé. Sur les résultats que nous avons obtenu, nous avons extrait les 50 premiers documents considérés comme les plus pertinents par notre fonction d'évaluation. Ensuite, nous avons construit la matrice de similarité de ces documents grâce à la fonction de similarité *cosine* (comme indiqué dans la section 8.2), puis nous avons exécuté l'algorithme de classification par nuage d'agents présenté dans ce chapitre. La figure 8.4 montre un exemple de l'interface de classification en cours d'analyse.

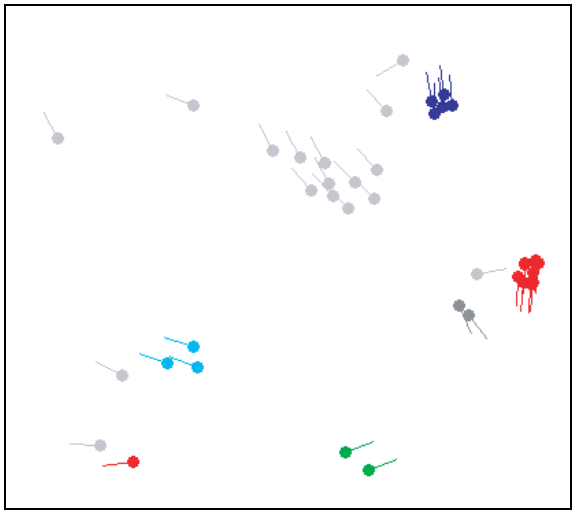


FIG. 8.4 – Visualisation obtenue par la méthode de classification de résultats du moteur de recherche par nuages d'agents. Chaque couleur représente une classe déterminée par l'algorithme de classification.

8.3.1 Première requête de test

Les premiers tests ont porté sur la requête 2 de la table 5.1. Cette requête traite des algorithmes génétiques et des algorithmes à base de fourmis artificielles. La liste des URL des documents résultats ainsi que les classes obtenues après regroupement des éléments similaires sont présentés en annexe dans la section A.1. Pour chaque classe trouvée, les liens sont disposés dans l'ordre de pertinence calculé par notre moteur de recherche. C'est-à-dire que le premier document d'une classe correspond au document le plus pertinent du groupe tandis que le dernier de la liste est considéré comme correspondant le moins à la requête.

On peut signaler que l'ensemble des résultats retournés paraît très similaire et traitent tous des algorithmes évolutionnaires et en particulier des algorithmes génétiques et à base d'agents. Il est donc à priori difficile d'établir une classification absolue de l'ensemble de ces éléments.

On constate dans un premier temps que 3 classes se sont formées. La première classe comprend la majorité des documents (26 éléments) tandis que les deux autres se partagent équitablement les effectifs restants. Une analyse rapide des documents retournés permet d'établir qu'une grande majorité correspond à une liste d'articles portant sur le sujet recherché : les algorithmes génétiques et les fourmis artificielles.

La première classe contient en majorité des fichiers ayant un caractère similaire : il s'agit de listes de publication (sous différents formats) traitant communément des algorithmes génétiques et des fourmis artificielles. Seuls 6 documents échappent à cette règle : les documents 7, 13, 20, 21, 23 et 26 contiennent du texte et ne ressemblent pas aux autres membres du groupe. La classification a toutefois regroupé des documents traitant objectivement du même sujet. On retrouve dans cette classe 6 pages Web ayant trait à la conférence EMOO (*Evolutionary MultiObjective Optimization*). Il s'agit des documents 4, 9, 10 ,11, 12 et 15. Seul un autre document provenant de cette conférence n'est pas regroupé dans la même classe : le document 7 du deuxième groupe.

On peut remarquer également que la classe 2 rassemble des documents particuliers ayant un format spécifique : des fichiers *bibtex* correspondant à des références bibliographiques utilisés en LaTeX. Dans le corpus étudié, seul un document de ce type échappe à ce regroupement et se situe dans la classe 1. Les autres éléments composant le groupe correspondent à quelques exceptions près - le document 7 dont nous avons parlé dans le paragraphe précédent et les documents 2, 3 et 4 - à des textes relativement petits traitant du sujet recherché. Des documents très similaires, tout du moins du point de vue de leur présentation et de leur contenu sont regroupés dans cette classe. Les documents 2 et 3 sont en effet très proches et proviennent tous deux de la base de données *Alife*. De même, les pages 9 et 10 sont similaires et se retrouvent côte à côte dans le même groupe. La classification a réussi à détecter les pages quasi identiques et les a regroupées dans les mêmes groupes.

Au contraire des deux classes précédemment évoquées, la troisième classe regroupe des documents contenant uniquement un article concernant le sujet recherché et non pas des listes d'articles accompagnés de leurs résumés. Seul le document 3 fait exception à cette règle. On peut également remarquer que le document 7 aurait dû être regroupé avec le document 23 de la classe 1. Il s'agit en effet d'un même article accompagné de commentaires différents, le document de la classe 3 comprenant beaucoup plus de liens bibliographiques.

Globalement, on peut dire que l'algorithme détecte bien les documents quasi identiques, même si leurs URL n'ont aucun point commun. C'est un atout non négligeable pour un moteur de recherche car cela permet de réduire le temps d'examen des résultats pour un utilisateur. Pour cette requête, la classification par nuages d'agents sépare

relativement bien les articles des listes d'articles et des références bibliographiques.

8.3.2 Deuxième exemple

Dans un deuxième temps, nous avons effectué des tests sur la requête 3 de la table 5.1. Le protocole expérimental reste identique à celui utilisé dans la section précédente. Cette requête traite de la technologie des fibres optiques et évoque la crise que ce secteur a connue récemment. La liste des URL des documents résultats ainsi que les classes obtenus après regroupement des éléments similaires sont présentés en annexe dans la section A.2. On retrouve ici 4 classes comportant respectivement 11, 29, 7 et 3 éléments.

Le premier groupe de documents rassemble des pages de cours ou d'explications des technologies engendrées par la fibre optique. Ces documents abordent l'aspect technique et physique de la fibre optique. Seuls les documents 5 et 6 de la classe 2 peuvent être considérés mal classés et auraient dû faire partie de ce groupe d'éléments.

La classe 2 regroupe un ensemble de sites commercialisant des technologies à base de fibre optique voire des fibres optiques elles-mêmes. On note toutefois l'exception des documents 5 et 6 évoqués dans le paragraphe précédent. Il est également intéressant de remarquer qu'un ensemble de documents très similaires ont été regroupés. Il s'agit des documents 9 à 29 de cette classe. Ils sont tous issus du même site proposant la vente d'éléments intervenant dans la mise en place d'une fibre optique. Le format de ces pages est bien évidemment très similaire ce qui explique le regroupement de ces éléments. Cela permet à l'utilisateur du moteur de recherche de ne pas perdre son temps à trier lui-même ces éléments si l'information qu'il recherche n'y est pas comprise.

Dans la troisième classe sont regroupées des pages très similaires entre elles. Elles proviennent en effet d'un même site. Il s'agit des pages de recensement de liens décrivant des technologies liées à la fibre optique d'un producteur de fibre optique. Le regroupement de telles pages est de bon augure pour l'utilisateur qui n'aura à examiner qu'un membre du groupe pour déterminer si ce type de page lui convient ou non.

La dernière classe regroupe 3 documents traitant exclusivement de conférences sur le thème des technologies à base de fibres optiques. Le vocabulaire employé dans ce type de documents est spécifique, la classification en a, par conséquent, fait un groupe à part.

Cet exemple montre l'efficacité et l'intérêt premier d'une classification de résultats d'un moteur de recherche. L'algorithme a séparé de manière très judicieuse quatre types de documents permettant à un utilisateur d'accéder plus rapidement à l'information qu'il recherche. On peut considérer que très peu d'erreurs de classification ont été commises : seuls deux documents paraissent mal classés sur l'ensemble des pages Web prises en compte.

8.3.3 Dernier exemple

Afin de valider les remarques faites dans les sections précédentes, nous avons analysé une dernière classification de résultats du moteur de recherche. Les tests sont cette fois-ci

effectués sur la requête 8 de la table 5.1. Le protocole expérimental est bien évidemment toujours le même et traite les 50 premiers résultats issus de la recherche. Le sujet de la requête porte sur des éléments spécifiques de programmation informatique : les *dll* et les *templates*. La liste des URL des documents résultats ainsi que les classes obtenues après regroupement des éléments similaires sont présentés en annexe dans la section A.3.

Cette fois-ci, 6 classes ont étés déterminées. La classe 2 regroupe la majorité des individus en comptabilisant 27 documents. Les classes 1, 3, 4, 5 et 6 regroupent respectivement 5, 10, 3, 2 et 3 éléments.

La première classe rassemble principalement des archives de la liste de diffusion du logiciel *pipermail* suivant plusieurs périodes. Les deux premiers documents sont strictement identiques mais apparaissent sous des noms de domaines différents. Encore une fois, l'algorithme de classification reconnaît bien les mêmes documents apparaissant sous des adresses différentes. Le document 4 de ce groupe correspond à la description des fonctions d'une bibliothèque : *Ogre*. Le dernier document contient une longue liste de termes de recherche associés au thème des *dll*. Il n'existe pas a priori de raison pour que ces documents soient classés ensemble.

La deuxième classe contient le plus grand nombre de documents. Ils se caractérisent cependant tous par leur longueur et traitent de sujets variés de l'informatique qui restent généraux. Aucun de ces documents ne traite d'un sujet spécifique et il paraît naturel de les regrouper. Les éléments identiques sont également regroupés dans la même classe comme par exemple les documents 2 et 3 ou encore les documents 20, 21 et 23.

La troisième classe paraît plus intéressante. Elle regroupe l'ensemble des documents parlant d'un sujet particulier de programmation : l'interface *ole* et *atl* ainsi que leur utilisation dans les bases de données. Seul le document 7 n'évoque pas ce sujet, mais il contient une liste importante de fichiers exécutables, ainsi que leur description, utilisés dans la gestion de ces mécanismes. Il est par conséquent compréhensible que l'algorithme de classification l'ait considéré comme similaire aux autres éléments du groupe.

La classe suivante comprend des documents listant les changements opérés au sein de différentes versions d'un programme. La classe 5 ne comprend que deux documents dans lesquels sont listés des erreurs détectés dans des programmes particuliers. On peut signaler que le document 2 de la classe 4 aurait pu être intégré facilement à cette classe. Il n'aurait pas été choquant non plus de regrouper ces deux classes afin de former un unique groupe de 5 éléments.

Enfin, la dernière classe regroupe 3 documents particuliers traitant du même sujet. Il s'agit d'un article sur les problèmes rencontrés lors de la programmation de *dll*. Ces trois documents diffèrent seulement par des commentaires disposés à la suite de l'article.

8.4 Conclusion

Le regroupement des résultats d'un moteur de recherche par similarité est très intéressant du point de vue de l'utilisateur. Dans les tests que nous avons réalisés dans le chapitre 7, les utilisateurs nous ont fait remarquer que trop de documents se ressem-

blant étaient retournés en résultat d'une recherche. La classification permet de dégager des groupes traitant du même sujet et en quelque sorte de prévenir l'utilisateur que ces documents sont susceptibles de se ressembler fortement.

Comme nous avons pu le constater dans la section précédente, les résultats obtenus par la classification par nuages d'agents en utilisant la mesure de similarité *cosine* apporte de très bons résultats. Elle peut également être utilisée afin de déterminer des documents apparaissant sous des adresses différentes ou qui ne différent que de quelques mots seulement. Cela permet d'éviter d'obtenir des doublons dans les résultats proposés à l'utilisateur.

Une fois la matrice de similarité calculée, le temps et les ressources requises pour exécuter l'algorithme de classification sont très faibles. L'utilisateur peut par conséquent exécuter la classification sur son navigateur afin d'observer l'évolution de la formation des classes. Il peut également interagir directement sur le déroulement de l'algorithme, par exemple en le stoppant prématurément. Cela permet d'offrir une information supplémentaire à l'utilisateur sur les résultats qui lui sont présentés sans pour autant les rendre plus complexe à interpréter.

Nous n'avons pas pris en compte dans cette adaptation de l'algorithme de classification les aspects interactifs avec l'utilisateur. À l'instar de *Squid* présenté dans la section 1.4.3 du chapitre 1 nous pouvons permettre de figer le déplacement des agents et de les manipuler pour les déplacer à un autre endroit de l'espace. Cette technique permet d'accélérer de classification ou de modifier les résultats obtenus. C'est une des suites directes de ce travail.

Conclusion

Bilan du travail réalisé

Dans ce travail de thèse, nous avons étudié le problème de la recherche d'information sur Internet. Nous avons proposé une nouvelle approche pour la résolution de ce problème complémentaire à celle privilégiée aujourd'hui dans les moteurs de recherche.

Dans le premier chapitre, nous avons établi un état de l'art décrivant les composantes essentielles d'un moteur de recherche : les méthodes de formulation d'une requête, les algorithmes de détection de pertinence et les principes de visualisation utilisés pour mettre en valeur les résultats. Ces éléments nous permettent de fonder une nouvelle stratégie de recherche et d'identifier les principes fondateurs de notre outil de recherche d'information. En particulier, cela nous a permis d'identifier les points sur lesquels se concentre ce travail de thèse : la formulation de la requête, la stratégie de recherche et la présentation des résultats.

Nous avons par conséquent défini un modèle s'appuyant sur la transformation du problème original en un problème d'optimisation. Pour cela, Internet est vu comme un graphe dans lequel les nœuds représentent les documents et les arcs orientés les liens hypertextes existant entre les pages Web. Nous définissons de plus une fonction d'évaluation s'appliquant à chaque nœud du graphe et donc à chaque document. L'association de ces deux propriétés nous a permis de construire un espace de recherche mathématique dans lequel chaque point possède un voisinage et peut être évalué. Le problème se résume alors à trouver les points maximisant la fonction d'évaluation dans l'espace de recherche ainsi construit, définition que l'on retrouve classiquement en optimisation.

Pour parcourir un espace de recherche, tout algorithme d'optimisation utilise des opérateurs. Ces derniers peuvent être au moins de deux natures différentes : générer de nouveaux points valides dans l'espace de recherche à partir de rien et déterminer un point à analyser dans le voisinage d'un autre point de l'espace de recherche. Le premier opérateur peut être réalisé en interrogeant les moteurs de recherche classiques afin de récupérer les résultats qu'ils fournissent et le deuxième correspond à une exploration locale en optimisation et consiste à suivre les liens hypertextes dans le cas du Web.

Nous nous sommes ensuite inspirés de techniques éprouvées en optimisation pour élaborer des stratégies de recherche efficaces. Nous avons détaillé dans le chapitre 2 les

principes généraux sur lesquels reposent les algorithmes génétiques ainsi que les approches à base de fourmis artificielles et les méthodes tabou. Quatre méta-heuristiques adaptées à notre problème ont alors vu le jour : *Geniminer* et *GeniminerII* inspirées des AG et décrites dans les chapitres 4 et 5, *Antsearch* utilisant des fourmis artificielles et *Tabusearch* utilisant des méthodes tabou décrites toutes les deux dans le chapitre 6.

Afin d'accélérer le traitement nécessaire à la mise en œuvre de nos méthodes et d'exploiter au mieux les ressources mises à notre disposition, nous avons conçu un modèle capable de distribuer l'effort de recherche à la fois au niveau logiciel et matériel. Les trois dernières méta-heuristiques exploitent une parallélisation logicielle permettant d'évaluer simultanément un grand nombre de documents. Ces analyses peuvent alors être réalisées sur une seule machine en utilisant des threads d'exécution ou réparties sur un réseau de machines connectées sur un serveur répartissant les demandes issues du moteur de recherche. Cette parallélisation a prouvé son efficacité.

Nous avons évalué les différents algorithmes de recherche que nous avons conçus dans le chapitre 7 en les comparant à un méta-moteur compilant les résultats issus des moteurs de recherche classiques grâce à la fonction d'évaluation que nous avons défini. Les résultats montrent que la méthode génétique (*GeniminerII*) surpasse les autres méthodes de recherche inspirées des heuristiques d'optimisation. Cette méthode obtient également des résultats comparables au méta-moteur utilisant notre fonction d'évaluation. L'évaluation de *GeniminerII* par des experts en comparaison avec les résultats issus des moteurs de recherche montre que notre méthode apporte une réelle amélioration des algorithmes de recherche existants. Dès lors que l'utilisateur spécifie une requête avec beaucoup de détails, notre approche génétique fournit majoritairement de meilleures réponses.

Le dernier point que nous abordons dans le chapitre 8 de cette thèse concerne la présentation et la visualisation des résultats. Les différentes études que nous avons analysées dans le chapitre 1 ainsi que les remarques faites par les utilisateurs de notre système de recherche nous ont amené à utiliser une classification des documents établis en tant que résultats. Ce regroupement permet de considérer des documents similaires comme des ensembles facilitant d'autant plus le tri pour l'utilisateur. Les résultats obtenus montrent que le regroupement de documents correspond à une réalité et qu'il facilite la lecture des résultats de notre moteur de recherche.

Perspectives

Les perspectives issues de ce travail sont diverses et touchent un grand nombre d'éléments composant un système de RI. Nous avons mis en œuvre dans notre approche une requête plus complète que celle utilisée dans les moteurs de recherche actuels. Il paraît judicieux de continuer dans cette démarche en proposant de nouveaux critères. Par exemple, un utilisateur pourrait spécifier un ensemble de documents dans la requête,

le système de recherche se chargeant alors de retrouver des documents similaires en réponse. Ce type de système peut également permettre de reformuler une requête en fonction des résultats renvoyés par une précédente exécution du moteur de recherche (*relevance feedback*). Un utilisateur sélectionnerait les documents ayant le mieux répondu à la question afin de préciser d'avantage son souhait.

Il serait également intéressant de développer l'aspect de veille stratégique : un système de capitalisation de connaissances d'un domaine particulier, accumulant les résultats issus de plusieurs requêtes, peut apporter des informations importantes sur l'évolution du traitement de ce domaine. Cette méthode peut être couplée à un système d'émission d'alertes lorsque de nouvelles informations sont prêtes à être mises à disposition des utilisateurs.

À l'instar de certains méta-moteurs, on peut imaginer ajouter une fonction de filtrage collaboratif à notre outil de recherche. De cette manière, les utilisateurs du système peuvent souligner les pages qui leur paraissent les plus intéressantes et partager leurs évaluations avec le reste de la communauté. Les résultats sélectionnés par les utilisateurs peuvent être considérés comme très pertinents et servir de base à la requête comme décrit au début de cette section.

Nous avons abordé à la fin de cette thèse la problématique de la visualisation de résultats d'un moteur de recherche. La méthode que nous avons mise en œuvre, bien qu'efficace, peut être rendue interactive afin d'impliquer d'avantage encore l'utilisateur dans l'interprétation des résultats. De plus, on constate que l'approche à base de nuages a l'avantage de trouver très vite des classes au sein des données. Cependant, son aspect dynamique et de mouvement "perpétuel" n'est pas forcément utile pour présenter les résultats du moteur, et trouver un bon critère d'arrêt n'est pas forcément facile. Nous souhaitons donc réaliser une hybridation de notre algorithme de catégorisation avec un affichage de graphe à base de forces et de ressorts, ce dernier algorithme souffrant d'un problème d'initialisation des objets à afficher mais ayant l'avantage de bien stabiliser les objets. Cette hybridation aurait pour but de terminer convenablement l'exécution du nuage d'agents afin d'atteindre un état "fixe" qui soit le plus informatif possible pour l'utilisateur.

Enfin, nous proposons dans notre modélisation une distribution avancée de l'effort de recherche sur un grand nombre de machines. Nous n'avons cependant pas pu tester cette approche à l'échelle du réseau Internet. En proposant à n'importe quelle personne ayant accès à nos serveurs depuis le Web d'offrir le temps processeur non utilisé de son ordinateur pour effectuer le téléchargement et l'analyse des documents requis par notre système, nous obtiendrons une application massivement parallèle. Cette architecture distribuée permettra alors de proposer une solution de recherche innovante accessible par un nombre conséquent d'utilisateurs.

Bibliographie

[Abramson et Abela, 1991] D. Abramson et J. Abela. A parallel genetic algorithm for solving the school timetabling problem. Technical report, C.S.I.R.O., 1991.

[Albert *et al.*, 1999] R. Albert, H. Jeong, et A.-L. Barabasi. Diameter of the world wide web. *Nature*, 401 :130–131, 1999.

[Allen *et al.*, 1993] Robert B. Allen, Pascal Obry, et Michael Littman. An interface for navigating clustered document sets returned by queries. In *Proceedings of the conference on Organizational computing systems*, pages 166–171. ACM Press, 1993.

[Altavista] Moteur de recherche Altavista, http ://www.altavista.com.

[Aoki, 1984] I. Aoki. Internal dynamics of fish schools in relation to inter-fish distance. *In Bulletin of the Japonese Society of Scientific Fisheries*, 50(5) :751–758, 1984.

[Arocena *et al.*, 1997] Gustavo O. Arocena, Alberto O. Mendelzon, et George A. Mihaila. Applications of a Web query language. *Computer Networks and ISDN Systems*, 29(8–13) :1305–1315, 1997.

[Azzag *et al.*, 2004] Hanene Azzag, Fabien Picarougne, Christiane Guinot, et Gilles Venturini. Un survol des algorithmes biomimétiques pour la classification. In M. Chavent et M. Langlais, editor, *Classification et fouille de données*, volume RNTI-C-1 of *RNTI*, pages 13–24. Cepadues, 2004.

[Bachelet *et al.*, 1998] V. Bachelet, Z. Hafidi, Ph. Preux, et E-G. Talbi. Vers la coopération de métaheuristiques parallèles. In *Calculateurs Parallèles*, volume 10(2) of *Réseaux et Systèmes Répartis*, pages 211–223, Avril 1998.

[Baeza-Yates et Ribeiro-Neto, 1999] Ricardo A. Baeza-Yates et Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[Baluja, 1994] Shumeet Baluja. Population-based incremental learning : A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1994.

[Barabási *et al.*, 2000] Albert-László Barabási, Réka Albert, et Hawoong Jeong. Scale-free characteristics of random networks : the topology of the World Wide Web. *Physica A : Statistical Mechanics and its Applications*, 281 :69–77, 2000.

[Battiti et Tecchiolli, 1994] R. Battiti et G. Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2), 1994.

[Battitti et Tecchiolli, 1992] R. Battitti et G. Tecchiolli. Parallel based search for combinatorial optimization - genetic algorithms and tabu. *Microprocessors and Micro-Systems*, 16(7) :351–367, 1992.

[Belding, 1995] Theodore C. Belding. The distributed genetic algorithm revisited. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 114–121. Morgan Kaufmann Publishers Inc., 1995.

[Bellot, 2000] Patrice Bellot. *Méthodes de classification et de segmentation locales non supervisées pour la recherche documentaire*. PhD thesis, Université d'Avignon et des Pays de Vaucluse, 2000.

[Bharat et Henzinger, 1998] Krishna Bharat et Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU, 1998.

[Blake et Merz, 1998] C.L. Blake et C.J. Merz. UCI repository of machine learning databases, 1998.

[Bonabeau *et al.*, 1999] E. Bonabeau, M. Dorigo, et G. Theraulaz. *Swarm Intelligence : From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[Borodin *et al.*, 2001] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, et Panayiotis Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *Proceedings of the tenth international conference on World Wide Web*, pages 415–429. ACM Press, 2001.

[Breder, 1951] C.M. Breder. Studies in the structure of the fish school. *Bull. Am. Museum Nat. Hist.*, 98(7FF), 1951.

[Breder, 1959] C.M. Breder. Studies in the social grouping of fishes. *Bull. Am. Museum Nat. Hist.*, 117(399ff), 1959.

[Brin *et al.*, 1998] Sergey Brin, Rajeev Motwani, Lawrence Page, et Terry Winograd. What can you do with a web in your pocket ? *Data Engineering Bulletin*, 21(2) :37–47, 1998.

[Brin et Page, 1998] Sergey Brin et Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7) :107–117, 1998.

[Broder *et al.*, 2000] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, et Janet Wiener. Graph structure in the web. In *Proceedings of the Ninth International World Wide Web Conference*. Elsevier, 2000.

[Calvert *et al.*, 1997] Kenneth L. Calvert, Matthew B. Doar, et Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6) :160–163, June 1997.

[Camazine *et al.*, 2001] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, et Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[Cantú-Paz, 1999] Erick Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[Cantú-Paz, 2000] Erick Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[Carey *et al.*, 2000] M. Carey, F. Kriwaczek, et S.M. Rüger. A visualization interface for document searching and browsing. In *Proceedings of CIKM 2000 Workshop on New Paradigms in Information Visualization and Manipulation*, 2000.

[Carey et Heesch, 2003] M. Carey et S.M. Heesch, D.and Rüger. Info navigator : A visualization tool for document searching and browsing. In *Proceedings of the 9th International Conference on Distributed Multimedia Systems (DMS'2003)*, 2003.

[Chakrabarti *et al.*, 1998a] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, et S. Rajagopalan. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*, pages 65–74, 1998.

[Chakrabarti *et al.*, 1998b] Soumen Chakrabarti, Byron Dom, Rakesh Agrawal, et Prabhakar Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal : Very Large Data Bases*, 7(3) :163–178, 1998.

[Chakrabarti *et al.*, 1999] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan, et A. Tomkins. Hypersearching the web. *Scientific American*, 280 :54–60, June 1999.

[Chakrapani et Skorin-Kapov, 1993] Jaishankar Chakrapani et Jadranka Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Ann. Oper. Res.*, 41(1-4) :327–341, 1993.

[Chalmers, 1993] Matthew Chalmers. Using a landscape metaphor to represent a corpus of documents. In *Proc. European Conference on Spatial Information Theory*, volume 716 of *LNCS*, pages 377–390, 1993.

[Chekuri *et al.*, 1996] C. Chekuri, M. Goldwasser, Prabhakar Raghavan, et E. Upfal. Web search using automatic classification. In *Proceedings of WWW-96, 6th International Conference on the World Wide Web*, San Jose, US, 1996.

[Chen et Dumais, 2000] Hao Chen et Susan Dumais. Bringing order to the web : automatically categorizing search results. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–152. ACM Press, 2000.

[Chen, 1995] Hsinchun Chen. Machine learning for information retrieval : neural networks, symbolic learning, and genetic algorithms. *J. Am. Soc. Inf. Sci.*, 46(3) :194–216, 1995.

[Chimera et Shneiderman, 1994] Richard Chimera et Ben Shneiderman. An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents. *ACM Trans. Inf. Syst.*, 12(4) :383–406, 1994.

[Chirita *et al.*, 2003] Paul Alexandru Chirita, Daniel Olmedilla, et Wolfgang Nejdl. Finding related hubs and authorities. In *Proceedings of the 1st Latin-American Web Congress*, Santiago, Chile, 2003.

[Cho *et al.*, 1998] Junghoo Cho, Hector García-Molina, et Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7) :161–172, 1998.

[Cohn et Chang, 2000] David Cohn et Huan Chang. Learning to probabilistically identify authoritative documents. In *Proc. 17th International Conf. on Machine Learning*, pages 167–174. Morgan Kaufmann, San Francisco, CA, 2000.

[Cohoon *et al.*, 1987] J. P. Cohoon, S. U. Hedge, W. N. Martin, et D. Richards. Punctuated equilibria : A parallel genetic algorithm. In John J. Grefenstette, editor, *Genetic algorithms and their applications : Proc. of the second Int. Conf. on Genetic Algorithms*, pages 148–154, Hillsdale, NJ, 1987. Lawrence Erlbaum Assoc.

[Colorni *et al.*, 1991] A. Colorni, M. Dorigo, et V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, pages 134–142, 1991.

[Crainic *et al.*, 1995] Teodor Gabriel Crainic, Michel Toulouse, et Michel Gendreau. Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spektrum*, 17(2) :113–123, 1995.

[Crainic *et al.*, 1997] Teodor Gabriel Crainic, Michel Toulouse, et Michel Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1) :61–72, 1997.

[Crainic et Toulouse, 1998] Teodor Gabriel Crainic et Michel Toulouse. *Parallel Strategies for Metaheuristics*, chapter Parallel Metaheuristics, pages 205–251. Kluwer Academic, 1998.

[Crainic et Toulouse, 2003] Teodor Gabriel Crainic et Michel Toulouse. *State-of-the-Art Handbook in Metaheuristics*, chapter Parallel Strategies for Metaheuristics. Springer, 2003.

[Cucchiara, 1993] R. Cucchiara. Analysis and comparison of different genetic models for the clustering problem in image analysis. In R.F. Albrecht, C.R. Reeves, et N.C. Steele, editors, *International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 423–427. Springer-Verlag, 1993.

[Cugini *et al.*, 1996] J. Cugini, C. Piatko, et S. Laskowski. Interactive 3d visualization for document retrieval. In *Proceedings of the Workshop on New Paradigms in Information Visualization and Manipulation, ACM Conference on Information and Knowledge Management (CIKM '96)*, nov 1996.

[Cugini *et al.*, 2000] J. Cugini, S. Laskowski, et M. Sebrechts. Design of 3d visualization of search results : Evolution and evaluation. In *Proceedings of IST/SPIE's 12th Annual International Symposium : Electronic Imaging 2000 : Visual Data Exploration and Analysis (SPIE 2000)*, San Jose, CA, January 2000.

[Cugini, 2000] John Cugini. Presenting search results : Design, visualization, and evaluation. In *Presented at Information Doors – Where Information Search and Hypertext Link workshop (held in conjunction with the ACM Hypertext and Digital Libraries conferences)*, May 2000.

[Cutting *et al.*, 1992] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, et John W. Tukey. Scatter/gather : a cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329. ACM Press, 1992.

[Cvetkovič *et al.*, 1979] D.M. Cvetkovič, M. Boob, et H. Sachs. *Spectra of Graphs*. Academic press, 1979.

[Darwin, 1859] C. Darwin. *The Origin of Species by Means of Natural Selection*. Mentor Reprint, 1958, NY, 1859.

[Davis, 1975] J.M. Davis. Socially induced flight reactions in pigeons. *Animal Behaviour*, 23 :597–601, 1975.

[Davis, 1980] J.M. Davis. The cordinated aerobatics of dunlin flocks. *Animal Behaviour*, 28 :668–673, 1980.

[Davis, 1991] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, New York, 1991.

[Deb et Kalyanmoy, 2001] Kalyanmoy Deb et Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., 2001.

[Décrypthon] Programme Décrypthon, http ://www.infobiogen.fr/services/decrypthon.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, et D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JRSSB*, 39 :1–38, 1977.

[Dorigo et Gambardella, 1997] M. Dorigo et L.M. Gambardella. Ant colonies for the Traveling Salesman Problem. *BioSystems*, 43 :73–81, 1997.

[Dorigo, 1992] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. In Italian.

[Drori et Alon, 2002] Offer Drori et Nir Alon. Using documents classification for displaying search results list. Technical Report 46, Leibniz Center for Research in Computer Science, Hebrew University, Jerusalem, Israel, July 2002.

[Drori, 2000] Offer Drori. The benefits of displaying additional internal document information on textual database search result lists. *Lecture Notes in Computer Science*, 1923 :69–82, 2000.

[Erman *et al.*, 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, et D. Raj Reddy. The hearsay-ii speech-understanding system : Integrating knowledge to resolve uncertainty. *ACM Comput. Surv.*, 12(2) :213–253, 1980.

[Exalead] Moteur de recherche Exalead, http ://www.exalead.com.

[Falkenauer, 1994] E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2) :123–144, 1994.

[Faloutsos *et al.*, 1999] Michalis Faloutsos, Petros Faloutsos, et Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[Fan *et al.*, 1999] W. Fan, M. D. Gordon, et P. Pathak. Automatic generation of matching functions by genetic programming for effective information retrieval. In *Proceedings of the 1999 Americas Conference on Information Systems*, pages 49–51, Milwaukee, WI, USA, August 13-15 1999.

[Ferber, 1995] Jacques Ferber. *Les systemes multi-agents, vers une intelligence collective*. InterEditions, Paris, 1995.

[Fogarty et Huang, 1991] Terence C. Fogarty et Runhe Huang. Implementing the genetic algorithm on transputer based parallel processing systems. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 145–149. Springer-Verlag, 1991.

[Fogel *et al.*, 1966] L.J. Fogel, A.J. Owens, et M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, J., Chichester, UK, 1966.

[Fresneau, 1994] D. Fresneau. *Biologie et comportement social d'une fourmi ponérine néotropicale (Pachycondyla apicalis)*. Thèse d'état, Université de Paris XIII, Laboratoire d'Ethologie Expérimentale et Comparée, France, 1994.

[Gambardella et Dorigo, 1995] L.M. Gambardella et M. Dorigo. Ant-Q : A reinforcement learning approach to the Travelling Salesman Problem. In A. Prieditis et S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 252–260. Morgan Kaufmann, San Mateo, California, 1995.

[Garey et Johnson, 1979] Michael R. Garey et David S. Johnson. *Computers and Intractability : A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, California, 1979.

[Glover et Laguna, 1993] Fred Glover et M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

[Glover et Laguna, 1998] Fred Glover et Fred Laguna. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[Glover, 1986] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5) :533–549, 1986.

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

[Google] Moteur de recherche Google, http ://www.google.com.

[Goss *et al.*, 1989] S. Goss, D. Fresneau, J.L. Deneubourg, J.P. Lachaud, et J. Valenzuela-Gonzalez. Individual foraging in the ant *pachycondyla apicalis*. *Œcologia*, 80 :65–69, 1989.

[Grefenstette, 1981] John Grefenstette. parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Computer Science Department, Carnegie Mellon University, Vanderbilt University, Nashville, TN, 1981.

[Harman, 1993] Donna K. Harman. The first text retrieval conference (trec-1) rockville, md, u.s.a., 4-6 november, 1992. *Inf. Process. Manage.*, 29(4) :411–414, 1993.

[Hauser et Männer, 1994] Rainer Hauser et Reinhard Männer. Implementation of standard genetic algorithm on mimd machines. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 504–513. Springer-Verlag, 1994.

[Hearst et Pedersen, 1996] Marti A. Hearst et Jan O. Pedersen. Reexamining the cluster hypothesis : Scatter/gather on retrieval results. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, pages 76–84, Zürich, CH, 1996.

[Hearst, 1994] Martha Alice Hearst. *Context and Structure in Automated Full-text Information Access.* PhD thesis, University of California at Berkeley, 1994.

[Hearst, 1995] Martha Alice Hearst. Tilebars : Visualization of term distribution information in full text information access. In *Proceedings of the Conference on Human Factors in Computing Systems, CHI'95*, 1995.

[Hemmje *et al.*, 1994] Matthias Hemmje, Clemens Kunkel, et Alexander Willett. Lyberworld - a visualization user interface supporting fulltext retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 249–259. Springer-Verlag New York, Inc., 1994.

[Hewitt *et al.*, 1973] C. Hewitt, P. Bishop, et R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proc. of the 3rd IJCAI*, pages 235–245, Stanford, MA, 1973.

[Hewitt, 1977] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8 :323–364, 1977.

[Hoffman *et al.*, 1999] Patrick Hoffman, Georges Grinstein, et David Pinkney. Dimensional anchors : a graphic primitive for multidimensional multivariate information visualizations. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management*, pages 9–16. ACM Press, 1999.

[Holland, 1975] John H. Holland. *Adaptation in natural artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[Hölldobler et Wilson, 1990] Bert Hölldobler et Edward O. Wilson. *The Ants.* The Belknap Press Cambridge, CambMass, 1990.

[Jain et Dubes, 1988] Anil K. Jain et Richard C. Dubes. *Algorithms for clustering data.* Prentice-Hall, Inc., 1988.

[Johnson et Shneiderman, 1991] B. Johnson et B. Shneiderman. Tree-maps : A space-filling approach to the visualization of hierarchical information structures. In *Proc. of Visualization'91*, pages 284–291, San Diego, CA, 1991.

[Jones et Beltrano, 1991] D.R. Jones et M.A. Beltrano. Solving partitioning problems with genetic algorithms. In R.K. Belew et L.B. Booker, editors, *Proceedings of*

the Fourth International Conference on Genetic Algorithms, pages 442–449. Morgan Kaufmann, San Mateo, CA, 1991.

[Kartoo] Méta-moteur de rehcerche Kartoo, http ://www.kartoo.com.

[Kaugars, 1998] Karlis Kaugars. Integrated multi scale text retrieval visualization. In *CHI 98 conference summary on Human factors in computing systems*, pages 307–308. ACM Press, 1998.

[Kishi *et al.*, 2000] Nobuko Kishi, Takahiro Ohmori, Seiji Saszauka, Akiko Kondo, Masahiro Mizutani, et Takahide Ogawa. Estimating web properties by using search engines and random crawlers. In *Proceedings of the INET'00 Conference*, Yokohama, Japan, july 2000. Internet Society.

[Kleinberg *et al.*, 1999] Jon M. Kleinberg, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, et Janet Wiener. The web as a graph : measurements, models, and methods. In *Proceedings of the 5th International Conference on Computing and Combinatorics(COCOON)*, July 1999.

[Kleinberg, 1999] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5) :604–632, 1999.

[Kohonen, 1989] T. Kohonen. *Self-organization and associative memory : 3rd edition.* Springer-Verlag New York, Inc., 1989.

[Korfhage, 1991] Robert R. Korfhage. To see, or not to see - is that the query ? In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 134–141. ACM Press, 1991.

[Koza, 1992a] John R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[Koza, 1992b] John R. Koza. Hierarchical automatic function definition in genetic programming. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 297–318, Vail, Colorado, USA, 24–29 July 1992. Morgan Kaufmann.

[Kraft *et al.*, 1997] D. Kraft, F. E. Petry, B. P. Buckles, et T. Sadisavan. Genetic algorithms for query optimization in information retrieval : Relevance feedback. In L. A. Zadeh E. Sanchez, T. Shibata, editor, *Genetic algorithms and fuzzy logic systems : Soft computing perspectives*, pages 155–173. World Scientific, Singapore, 1997.

[Kumar *et al.*, 1999] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, et Andrew Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks (Amsterdam, Netherlands : 1999)*, 31(11–16) :1481–1493, 1999.

[Kumar *et al.*, 2000] S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, et Eli Upfal. The web as a graph. In *Proceedings of the Symposium on Principles of Database Systems*, pages 1–10, 2000.

[Kuntz *et al.*, 1997] P. Kuntz, P. Layzell, et D. Snyers. A colony of ant-like agents for partitioning in VLSI technology. In P. Husbands et I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life*, pages 417–424. MIT Press, Boston, 1997.

[Kwok *et al.*, 2001] Cody C. T. Kwok, Oren Etzioni, et Daniel S. Weld. Scaling question answering to the web. In *World Wide Web*, pages 150–161, 2001.

[Lamping *et al.*, 1995] John Lamping, Ramana Rao, et Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM Press/Addison-Wesley Publishing Co., 1995.

[Landrin-Schweitzer *et al.*, 2003] Yann Landrin-Schweitzer, Pierre Collet, Evelyne Lutton, et Thierry Prost. Introducing lateral thinking in search engines with interactive evolutionary algorithms. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 214–219. ACM Press, 2003.

[Lawrence et Giles, 1999] Steve Lawrence et C. Lee Giles. Accessibility of information on the web. *Nature*, 400(6740) :107–109, 1999.

[Lempel et Moran, 2000] R. Lempel et S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *j-COMP-NET-AMSTERDAM*, 33(1–6) :387–401, June 2000.

[Lin, 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3-4) :293–321, 1992.

[Luhn, 1958] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2 :159–165, 1958.

[Lukasiewicz, 1963] J. Lukasiewicz. *Elements of Mathematical Logic*. Pergamon Press, 1963.

[Lumer et Faieta, 1994] E.D. Lumer et B. Faieta. Diversity and adaptation in populations of clustering ants. In D. Cliff, P. Husbands, J.A. Meyer, et Stewart W., editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 501–508. MIT Press, Cambridge, Massachusetts, 1994.

[Mackinlay *et al.*, 1991] Jock D. Mackinlay, George G. Robertson, et Stuart K. Card. The perspective wall : detail and context smoothly integrated. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 173–176. ACM Press, 1991.

[Mann, 1999] Thomas M. Mann. Visualization of WWW-search results. In *DEXA Workshop*, pages 264–268, 1999.

[McCrickard et Kehoe, 1997] S. McCrickard et C. Kehoe. Visualizing search results using sqwid. In *Proceedings of the Sixth International World Wide Web Conference*, April 1997.

[Meñkov *et al.*, 2000] Vladimir Meñkov, David J.Neu, et QinShi. Antworld : A collaborative web search tool. *DCW proceedings*, 2000.

[Menczer et Monge, 1999] Filippo Menczer et Alvaro E. Monge. Scalable web search by adaptive online agents : an infospiders case study. In M. Klusch, editor, *Intelligent Information Agents : Agent-Based Information Discovery and Management on the Internet*, pages 323–347, Berlin, 1999. Springer.

[Menczer, 2003] Filippo Menczer. Complementing search engines with online web mining agents. *Decision Support Syst.*, 35(2) :195–212, 2003.

[Mendelzon et Rafiei, 2000] Alberto O. Mendelzon et Davood Rafiei. What do the neighbours think ? computing web page reputations. *IEEE Data Engineering Bulletin*, 23(3) :9–16, 2000.

[Michalewicz, 1996] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, 1996.

[Mihaila, 1996] George A. Mihaila. *WebSQL - an SQL-like query language for the world wide web.* PhD thesis, University of Toronto, 1996.

[Mladenic, 1998] Dunja Mladenic. Turning yahoo to automatic web-page classifier. In *European Conference on Artificial Intelligence*, pages 473–474, 1998.

[Monmarché *et al.*, 1999a] N. Monmarché, G. Nocent, M. Slimane, et G. Venturini. Imagine : a tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'99)*, volume 3, pages 640–645, Tokyo, Japan, October 12-15 1999.

[Monmarché *et al.*, 1999b] N. Monmarché, M. Slimane, et G. Venturini. On improving clustering in numerical databases with artificial ants. In D. Floreano, J.D. Nicoud, et F. Mondala, editors, *5th European Conference on Artificial Life (ECAL'99), Lecture Notes in Artificial Intelligence*, volume 1674, pages 626–635, Swiss Federal Institute of Technology, Lausanne, Switzerland, 13-17 September 1999. Springer-Verlag.

[Monmarché *et al.*, 2000] N. Monmarché, G. Venturini, et M. Slimane. On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems*, 16(8) :937–946, 2000.

[Monmarché, 2000] Nicolas Monmarché. *Algorithme de fourmis artificielles : applications à la classification et à l'optimisation.* Thèse de doctorat, Université de Tours, 2000.

[Morgan et Kilgour, 1996] J.J. Morgan et A.C. Kilgour. Personalising information retrieval using evolutionary modelling. In *Proceedings of PolyModel 16 : Applications of Artificial Intelligence*, pages 142–149, 1996.

[Moukas, 1997] A. Moukas. Amalthea : information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the Conference on Practical Applications of Agents and Multiagent Technology*, volume 11, pages 437–457, London, April 1997.

[Mühlenbein, 1991] Heinz Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 316–337. Morgan Kaufmann, San Mateo, CA, 1991.

[Mukherjea et Foley, 1995] Sougata Mukherjea et James D. Foley. Visualizing the World-Wide Web with the Navigational View Builder. *Computer Networks and ISDN Systems*, 27(6) :1075–1087, 1995.

[Mukherjea et Hara, 1999] Sougata Mukherjea et Toshinori Hara. Visualizing world-wide web search engine results. In *Proceedings of 1999 IEEE International Conference on Information Visualization*, pages 400–405, London, England, July 1999.

[Munzner et Burchard, 1995] Tamara Munzner et Paul Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *Proceedings of the first symposium on Virtual reality modeling language*, pages 33–38. ACM Press, 1995.

[Netcraft] Netcraft, surveillance de site web, http ://www.netcraft.com.

[Ng *et al.*, 2001] Andrew Y. Ng, Alice X. Zheng, et Michael I. Jordan. Stable algorithms for link analysis. In *Proceedings of the 24th Annual Intl. ACM SIGIR Conference*. ACM, 2001.

[Nick et Themis, 2001] Zacharis Z. Nick et Panayiotopoulos Themis. Web search using a genetic algorithm. *IEEE Internet Computing*, 5(2) :18–26, 2001.

[Nowell *et al.*, 1996] Lucy Terry Nowell, Robert K. France, Deborah Hix, Lenwood S. Heath, et Edward A. Fox. Visualizing search results : some alternatives to query-document similarity. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 67–75. ACM Press, 1996.

[Ogden *et al.*, 1998] William C. Ogden, Mark W. Davis, et Sean Rice. Document thumbnail visualization for rapid relevance judgments : When do they pay off ? In Ellen M. Voorhees et Donna K. Harman, editors, *Proceedings of TREC-7, 7th Text Retrieval Conference*. National Institute of Standards and Technology, Gaithersburg, US, nov 1998.

[Page *et al.*, 1998] Lawrence Page, Sergey Brin, Rajeev Motwani, et Terry Winograd. The pagerank citation ranking : Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[Pant et Menczer, 2002] Gautam Pant et Filippo Menczer. Myspiders : Evolve your own intelligent web crawlers. *Autonomous Agents and Multi-Agent Systems*, 5(2) :221–229, 2002.

[Pathak *et al.*, 2000] Praveen Pathak, Michael, Gordon, et Weiguo Fan. Effective information retrieval using genetic algorithms based matching function adaptation. In *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS)*, Hawaii, USA, 2000.

[Picarougne *et al.*, 2002a] Fabien Picarougne, Claudie Fruchet, Antoine Oliver, Nicolas Monmarché, et Gilles Venturini. Recherche d'information sur internet par algorithme génétique. In *Actes des quatrièmes journées nationales de la ROADEF*, pages 247–248, Paris, France, 20-22 février 2002.

[Picarougne *et al.*, 2002b] Fabien Picarougne, Claudie Fruchet, Antoine Oliver, Nicolas Monmarché, et Gilles Venturini. Web searching considered as a genetic optimization problem. In *Local Search Two Day Workshop*, London, UK, 16-17 April 2002.

[Picarougne *et al.*, 2002c] Fabien Picarougne, Nicolas Monmarché, Antoine Oliver, et Gilles Venturini. Geniminer : Web mining with a genetic based algorithm. In *Procee-*

dings of the IADIS International Conference WWW/Internet, pages 263–270, Lisbon, Portugal, november 13-15 2002.

[Picarougne *et al.*, 2002d] Fabien Picarougne, Nicolas Monmarché, Antoine Oliver, et Gilles Venturini. Web mining with a genetic algorithm. In *Eleventh International World Wide Web Conference*, Honolulu, Hawaii, 7-11 May 2002.

[Picarougne *et al.*, 2003a] Fabien Picarougne, Nicolas Monmarché, Antoine Oliver, et Gilles Venturini. Geniminer, un moteur de recherche génétique. In *Extraction et Gestion des Connaissances (EGC 2003)*, volume 17 - N°1-2-3 of *RSTI Série RIA-ECA*, pages 319–330, Lyon, 22 au 24 Janvier 2003. Hermes.

[Picarougne *et al.*, 2003b] Fabien Picarougne, Nicolas Monmarché, Mohamed Slimane, Gilles Venturini, et Christiane Guinot. Geniminer, un outil pour la veille stratégique. In *Journées Francophones de la Toile - JFT 2003*, pages 323–324, Tours, France, 30 juin - 2 juillet 2003.

[Picarougne *et al.*, 2003c] Fabien Picarougne, Nicolas Monmarché, Mohamed Slimane, Gilles Venturini, et Christiane Guinot. Two bio-inspired metaheuristics for information search on the web. In *Proceedings of the 6th International Conference on Artificial Evolution*, pages 422–434, Marseille, France, 27-30 octobre 2003.

[Picarougne *et al.*, 2004a] Fabien Picarougne, Hanene Azzag, Gilles Venturini, et Christiane Guinot. On data clustering with a flock of artificial agents. In *Proceedings of the ICTAI 2004*, Boca Raton, Florida, USA, 15-17 november 2004. to appear.

[Picarougne *et al.*, 2004b] Fabien Picarougne, Gilles Venturini, et Christiane Guinot. Un algorithme génétique parallèle pour la veille stratégique sur internet. In *VSST 2004*, Toulouse, France, 25-29 octobre 2004. à paraître.

[Porto et Ribeiro, 1995] S. C. S. Porto et C. C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *j-INT-J-HIGH-SPEED-COMPUTING*, 7(1) :45–71, 1995.

[Preux et Talbi, 1999] P. Preux et E-G. Talbi. Towards hybrid evolutionary algorithms. *International Transactions in Operational Research (ITOR)*, 6 :557–570, 1999.

[Proctor et Winter, 1998] Glenn Proctor et Chris Winter. Information flocking : Data visualisation in virtual worlds using emergent behaviours. *Lecture Notes in Computer Science*, 1434 :168–176, 1998.

[Rauber et Bina, 1999] Andreas Rauber et Harald Bina. A metaphor graphics based representation of digital libraries on the world wide web : Using the libViewer to make metadata visible. In *DEXA Workshop*, pages 286–290, 1999.

[Rego et Roucairol, 1992] C. Rego et C. Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In J.P. Kelly L.H. Osman, editor, *Meta-Heuristics : Theory and Applications*, pages 253–295, Dordrecht, Norwell, MA, 1992. Kluwer Academic Publishers.

[Robertson *et al.*, 1992] Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Aarron Gull, et Marianna Lau. Okapi at TREC. In *Text REtrieval Conference*, pages 21–30, 1992.

[Rocchio, 1971] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART retrieval system : Experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ : Prentice Hall, 1971.

[Rochat et Taillard, 1995] Yves Rochat et Éric D. Taillard. Probalistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1 :147 – 167, 1995.

[Salton *et al.*, 1975] G. Salton, C. S. Yang, et C. T. Yu. A theory of term importance in automatic text analysis. *journal of the American Society for Information Sciencet*, 26(1) :33–44, 1975.

[Salton *et al.*, 1983] Gerard Salton, Edward A. Fox, et Harry Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11) :1022–1036, 1983.

[Salton et Buckley, 1988] G. Salton et C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5) :513–523, 1988.

[Salton et McGill, 1983] Gerard Salton et Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, 1983.

[Salton, 1971] G. Salton. The smart retrieval system. experiment in automatic document processing. *Prentice Hall*, 1971.

[Sammon, 1969] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5) :401–409, 1969.

[Sanchez et Pierre, 1994] E. Sanchez et P. Pierre. Fuzzy logic and genetic algorithms in information retrieval. In *3rd International Conference on Fuzzy Logic, Neural Networks and Soft Computing*, 1994.

[Sander-Beuermann et Schomburg, 1998] Wolfgang Sander-Beuermann et Mario Schomburg. Internet information retrieval - the further development of meta-search engine technology. In *Proceedings of the INET'98 Conference*, Genève, Switzerland, July 1998.

[Schlierkamp-Voosen et Mühlenbein, 1994] Dirk Schlierkamp-Voosen et Heinz Mühlenbein. Strategy adaptation by competing subpopulations. In Yuval Davidor, Hans-Paul Schwefel, et Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 199–208, Berlin, 1994. Springer.

[Schwefel et Rudolph, 1995] Hans-Paul Schwefel et Gunter Rudolph. Contemporary evolution strategies. In *European Conference on Artificial Life*, pages 893–907, 1995.

[Sebrechts *et al.*, 1999] Marc M. Sebrechts, John V. Cugini, Sharon J. Laskowski, Joanna Vasilakis, et Michael S. Miller. Visualization of search results : a comparative evaluation of text, 2d, and 3d interfaces. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10. ACM Press, 1999.

[Selberg et Etzioni, 1995] E. Selberg et O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International World-Wide Web Conference*, Darmstadt, Germany, December 1995.

[Seti@Home] Programme Seti@home, http ://setiathome.ssl.berkeley.edu.

[Sheth, 1994] Beerud D. Sheth. A learning approach to personalized information filtering. Master's thesis, MIT Media Lab, January 1994.

[Shneiderman *et al.*, 2000] Ben Shneiderman, David Feldman, Anne Rose, et Xavier Ferré Grau. Visualizing digital library search results with categorical and hierarchical axes. In *Proceedings of DL-00, 5th ACM Digital Library Conference*, pages 57–66, San Antonio, US, 2000.

[Shneiderman, 1992] Ben Shneiderman. Tree visualization with tree-maps : A 2-D space-filling approach. *ACM Transactions on Graphics*, 11(1) :92–99, 1992.

[Siganos *et al.*, 2003] Georgos Siganos, Michalis Faloutsos, Petros Faloutsos, et Christos Faloutsos. Power laws and the as-level internet topology. *IEEE/ACM Trans. Netw.*, 11(4) :514–524, 2003.

[Silverstein *et al.*, 1998] Craig Silverstein, Monika Henzinger, Hannes Marais, et Michael Moricz. Analysis of a very large altavista query log. Technical Report 1998-014, Digital SRC, 1998. http ://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1998-014.html.

[Spears et De Jong, 1991] William M. Spears et Kenneth A. De Jong. An analysis of multi-point crossover. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 301–315, San Mateo, CA, 1991. Morgan Kaufmann.

[Spiessens et Manderick, 1991] Piet Spiessens et Bernard Manderick. A massively parallel genetic algorithm : Implementation and first analysis. In Rick Belew et Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 279–286, San Mateo, CA, 1991. Morgan Kaufman.

[Spink *et al.*, 2001] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, et Tefko Saracevic. Searching the web : the public and their queries. *J. Am. Soc. Inf. Sci. Technol.*, 52(3) :226–234, 2001.

[Stützle et Hoos, 1997] T. Stützle et H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ Ant System and local search for the Traveling Salesman Problem. In *Proceedings of the fourth International Conference on Evolutionary Computation*, pages 308–313. IEEE Press, 1997.

[Swan et Allan, 1998] Russell C. Swan et James Allan. Aspect windows, 3-d visualizations, and indirect comparisons of information retrieval systems. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 173–181. ACM Press, 1998.

[Syswerda, 1989] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9. Morgan Kaufmann Publishers Inc., 1989.

[Taillard, 1993] Eric Denis Taillard. *Recherches iteratives dirigees paralleles.* PhD thesis, École Polytechnique Fédérale de Lausanne, 1993.

[Tanese, 1987] Reiko Tanese. Parallel genetic algorithm for a hypercube. In John J. Grefenstette, editor, *Genetic algorithms and their applications : Proc. of the second*

Int. Conf. on Genetic Algorithms, pages 177–183, Hillsdale, NJ, 1987. Lawrence Erlbaum Assoc.

[Todd et Sen, 1997] David S. Todd et P. Sen. A multiple criteria genetic algorithm for containership loading. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 674–681, San Francisco, July 19–23 1997. Morgan Kaufmann.

[Vakali et Manolopoulos, 1999] A. Vakali et Y. Manolopoulos. Caching objects from heterogeneous information sources. In *Proceedings International Database Conference (IDC'99)*, Hong-Kong, July 1999.

[Vallin et Coello, 2003] Max Vallin et Juan Manuel Adan Coello. An agent for web information dissemination based on a genetic algorithm. In *IEEE International Conference on Systems, Man and Cybernetics - IEEE SMC'03*, pages 3834–3839, Hyatt Regency, Washington, D.C., USA, 5 - 8 October 2003. IEEE Press.

[Van Rijsbergen, 1979] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

[Veerasamy et Belkin, 1996] Aravindan Veerasamy et Nicholas J. Belkin. Evaluation of a tool for visualization of information retrieval results. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 85–92. ACM Press, 1996.

[Veerasamy et Heikes, 1997] Aravindan Veerasamy et Russell Heikes. Effectiveness of a graphical display of retrieval results. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 236–245. ACM Press, 1997.

[Venturini, 1997] Gilles Venturini. *Apport des algorithmes génétiques à l'apprentissage et à l'optimisation.* Habilitation à diriger les recherches, Université de Tours, 12 1997.

[von Laszewski, 1991] G. von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R.K. Belew et L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann, San Mateo, CA, 1991.

[Washburne, 1927] J. N. Washburne. An experimental study of various graphic, tabular and textual methods of presenting quantitative material. *Journal of Educational Psychology*, 18(6) :361–376, 1927.

[Watkins et Dayan, 1992] C. J. Watkins et P. Dayan. Q-learning. *Machine Learning*, 8(3/4) :279–292, 1992.

[Watkins, 1989] C. J. Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge university, 1989.

[Whitley et Hanson, 1989] Darrell Whitley et Thomas Hanson. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the third international conference on Genetic algorithms*, pages 391–396. Morgan Kaufmann Publishers Inc., 1989.

[Whitley et Starkweather, 1990] D. Whitley et T. Starkweather. Genitor II : A distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2 :189–214, 1990.

[Whitley, 1989] Darrell Whitley. The genitor algorithm and selection pressure : why rank-based allocation of reproductive trials is best. In *Proceedings of the third international conference on Genetic algorithms*, pages 116–121. Morgan Kaufmann Publishers Inc., 1989.

[Whitley, 1993] L. Darrell Whitley. Cellular genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 658. Morgan Kaufmann Publishers Inc., 1993.

[Wise *et al.*, 1995] J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, et V. Crow. Visualizing the non-visual : spatial analysis and interaction with information from text documents. In *Proceedings of the 1995 IEEE Symposium on Information Visualization*, page 51. IEEE Computer Society, 1995.

[Wise, 1999] James A. Wise. The ecological approach to text visualization. *J. Am. Soc. Inf. Sci.*, 50(13) :1224–1233, 1999.

[Yahoo] Moteur de recherche Yahoo !, http ://www.yahoo.com.

[Yook *et al.*, 2002] Soon-Hyung Yook, Hawoong Jeong, et Albert-Laszlo Barabasi. Modeling the internet's large-scale topology. *PNAS*, 99(21) :13382–13386, 2002.

[Zadeh, 1965] L.A. Zadeh. Fuzzy sets. *Information Control*, 8 :338–353, 1965.

[Zamir et Etzioni, 1998] Oren Zamir et Oren Etzioni. Web document clustering : a feasibility demonstration. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54. ACM Press, 1998.

[Zamir et Etzioni, 1999] Oren Zamir et Oren Etzioni. Grouper : a dynamic clustering interface to Web search results. *Computer Networks (Amsterdam, Netherlands : 1999)*, 31(11–16) :1361–1374, 1999.

[Zipf, 1949] George Kingsley Zipf. Human behaviour and the principle of least effort. *Addison-Wesley, Cambridge, Massachusetts*, 1949.

[Zitzler, 2001] E. Zitzler. Evolutionary algorithms for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, et T. Fogarty, editors, *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 19–26, Athens, Greece, 2001. International Center for Numerical Methods in Engineering (Cmine).

Annexe A

Résultats obtenus par la classification par nuages d'agents

A.1 Première classification

Résultats obtenus sur la requête 2 de la table 5.1 en page 138 après classification par la méthode des nuages d'agents décrite dans le chapitre 8.

Classe 1

1. http ://www.cs.bham.ac.uk/~wbl/biblio/gp-bibliography.ref
2. http ://www.cs.bham.ac.uk/~wbl/biblio/gp-bib-num.txt
3. http ://www.cs.bris.ac.uk/~kovacs/lcs/lcs.html
4. http ://www.lania.mx/~ccoello/EMOO/EMOOconferences.html
5. http ://www.pubs.asce.org/WWWsrchkwx.cgi ?Algorithms
6. http ://citeseer.ist.psu.edu/context/18084/0
7. http ://www.ece.msstate.edu/~hagler/May2001/14/Begin.htm
8. http ://perso.wanadoo.fr/patrick.davalan/Liens/liens_genetic.html
9. http ://neo.lcc.uma.es/emoo/EMOObib.html
10. http ://www.lania.mx/~ccoello/EMOO/EMOObib.html
11. http ://www.jeo.org/emo/EMOOconferences.html
12. http ://neo.lcc.uma.es/emoo/EMOOconferences.html
13. http ://www.antoptima.ch/en/methods.phtml
14. http ://www.cs.iastate.edu/~gannadm/Bib/gann.bib
15. http ://www.lania.mx/~ccoello/EMOO/EMOOjournals.html
16. http ://143.129.203.3/eume/MIC2003/html/abstracts.php
17. http ://minimum.inria.fr/PPSN2000/AcceptedPapers.html
18. http ://pacosy.informatik.uni-leipzig.de/pv/Personen/middendorf/Ameis_MMi.engl.html
19. http ://www.ppgia.pucpr.br/~alex/papers.html

20. http ://www.ise.nus.edu.sg/proceedings/apors2000/fullpapers/02-03-fp.htm
21. http ://www.usa-shops.com/shop/algorithmbooks/genetic-algorithm-books-5.html
22. http ://citeseer.ist.psu.edu/context/7366/0
23. http ://www.codeproject.com/cpp/GeneticandAntAlgorithms.asp ?df=100 &forumid=26096&exp=0&select=862212
24. http ://citeseer.ist.psu.edu/340269.html
25. http ://www.genetic-programming.org/gp98tutorialabsbio.html
26. http ://www.getcited.org/pub/103396997

Classe 2

1. http ://www.cs.bham.ac.uk/~wbl/ftp/biblio/gecco2001.bib
2. http ://www.aridolan.com/ad/adb/GA.html
3. http ://web.tiscali.it/vitaartificiale/aldb/GA.html
4. http ://citeseer.ist.psu.edu/context/2669/0
5. http ://macedonia.uom.gr/~jason/phdjason.bib
6. http ://gal4.ge.uiuc.edu80/GECCO-2002/gecco2002.bib
7. http ://www.jeo.org/emo/EMOObib.html
8. http ://www.densis.fee.unicamp.br/~moscato/memetic.bib
9. http ://www.well.com/~xanthian/link_pages/Programming/Paradigms/ GeneticAlgorithms.html
10. http ://www.well.com/user/xanthian/link_pages/Programming/Paradigms/ GeneticAlgorithms.html
11. http ://citeseer.ist.psu.edu/context/1263627/0
12. http ://www.cs.cinvestav.mx/~constraint/
13. http ://citeseer.ist.psu.edu/384106.html

Classe 3

1. http ://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html
2. http ://mailgate.supereva.it/comp/comp.ai.genetic/msg06648.html
3. http ://citeseer.ist.psu.edu/context/157928/0
4. http ://www.azstarnet.com/~scottmc/javanoginn.htm
5. http ://www.cs.nmsu.edu/~dcook/thesis/paper2.html
6. http ://www.cs.iastate.edu/~baojie/acad/current/bega/572project/bega-news.htm
7. http ://www.codeproject.com/cpp/GeneticandAntAlgorithms.asp ?df=100 &forumid=26096&exp=0&select=665949
8. http ://jamesthornton.com/cf/paper ?paper_id=44
9. http ://www.cs.ust.hk/~hpguo/publications.html
10. http ://encyclopedia.thefreedictionary.com/Ant%20colony%20optimisation
11. http ://citeseer.ist.psu.edu/459780.html

A.2 Deuxième classification

Résultats obtenus sur la requête 3 de la table 5.1 en page 138 après classification par la méthode des nuages d'agents décrite dans le chapitre 8.

Classe 1

1. http ://www.telebyteusa.com/dslprimer/dslfull.htm
2. http ://www.ece.umd.edu/ davis/optfib.html
3. http ://www.stfx.ca/people/kyoung/webpage/469/469-teams/group8/fiberpaper.html
4. http ://knowledgestorm.techtarget.com/ssynd/user_search.jsp
5. http ://www.lightwavetc.com/box14.html
6. http ://lightwavetc.com/box14.html
7. http ://www.arcelect.com/fibercable.htm
8. http ://www.garrenshay.com/draft%20final%20report.htm
9. http ://lightwavetc.com/box15.html
10. http ://www.fiberopticsupplies.net/fiber_optic_cable_assemblies_resources.html
11. http ://wido.indointernet.com/telecommunication/indek_telkom.htm

Classe 2

1. http ://www.ee.umd.edu/ davis/optfib.html
2. http ://www.mindbranch.com/listing/product/R2-460.html
3. http ://www.buscom.com/advmat/GB233.html
4. http ://ruchichowdhury.tripod.com/fiber_optic_communication.htm
5. http ://www.comtest.com/tutorials/fiber.html
6. http ://www.igigroup.com/pub/reportseries/fopub.html
7. http ://rodjon.caro.net/htdocs/TECHLINKS/Telecommunications.html
8. http ://business.vsnl.com/foservice/software.htm
9. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000055_0002_1
10. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000023_0002_1
11. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000022_0002_1
12. http ://bcc.ecnext.com/coms2/summary__0002_000113_000008_000003_0002_1
13. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000014_0002_1
14. http ://bcc.ecnext.com/coms2/summary__0002_000113_000007_000051_0002_1
15. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000030_0002_1
16. http ://bcc.ecnext.com/coms2/summary__0002_000113_000008_000001_0002_1
17. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000007_0002_1
18. http ://bcc.ecnext.com/coms2/summary__0002_000113_000001_000000_0002_1
19. http ://bcc.ecnext.com/coms2/summary__0002_000113_000008_000006_0002_1
20. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000029_0002_1

21. http ://bcc.ecnext.com/coms2/summary__0002_000113_000007_000078_0002_1
22. http ://bcc.ecnext.com/coms2/summary__0002_000113_000006_000006_0002_1
23. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000075_0002_1
24. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000008_0002_1
25. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000024_0002_1
26. http ://bcc.ecnext.com/coms2/summary__0002_000113_000005_000016_0002_1
27. http ://bcc.ecnext.com/coms2/summary__0002_000113_000003_000017_0002_1
28. http ://bcc.ecnext.com/coms2/summary__0002_000113_000007_000000_0002_1
29. http ://bcc.ecnext.com/coms2/summary__0002_000113_000000_000000_0002_1

Classe 3

1. http ://www.redferncomponents.com/index/fiber_bragg_grating.htm
2. http ://www.redferncomponents.com/index/fiber_lasers.htm
3. http ://www.redferncomponents.com/index/fiber_sensors.htm
4. http ://www.redferncomponents.com/index/fiber_optics.htm
5. http ://www.redferncomponents.com/index/fiber_optic_manufacturing.htm
6. http ://www.redferncomponents.com/index/fiber_optic_sensors.htm
7. http ://www.redferncomponents.com/index/fiber_optic_components.htm

Classe 4

1. http ://www.spie.org/web/meetings/programs/pe99/confs/3860.html
2. http ://www.lightwavetc.com/box15.html
3. http ://www.spie.org/web/abstracts/2000/2070.html

A.3 Troisième classification

Résultats obtenus sur la requête 8 de la table 5.1 en page 138 après classification par la méthode des nuages d'agents décrite dans le chapitre 8.

Classe 1

1. http ://lists.gis.umn.edu/pipermail/mapserver-users/2003-July.txt
2. http ://lists.dmsolutions.ca/pipermail/mapserver-users/2003-July.txt
3. http ://lists.dmsolutions.ca/pipermail/mapserver-users/2002-September.txt
4. http ://www.ogre3d.org/docs/api/html/namespaceOgre.html
5. http ://www.technolopedia.com/dll.htm

Classe 2

1. http ://www.tomruby.com/clarion/alpha.html
2. http ://www.esrf.fr/computing/cs/taco/doc/manual/
3. http ://www.esrf.fr/computing/cs/taco/doc/manual/TACO.html
4. http ://www.iem.pw.edu.pl/ftp/listy/ntsec
5. http ://www.capesoft.com/docs/Replicate/Replicate.htm
6. http ://www.u-bourgogne.fr/CRI-CCUB/archives/faq-afuu/object
7. http ://www.ictp.trieste.it/~radionet/nuc1996/ref/winsock/wsspi22.htm
8. http ://www.os2bbs.com/Download/tools.html
9. http ://www.trumphurst.com/cpplibs/datapage.phtml
10. http ://www.softvelocity.net/faq/idxlist.htm
11. http ://www.clarionmag.com/cmag/news.html ?year=2003&limit=500&
12. http ://www.supelec-rennes.fr/ren/perso/lheye/nt/faqcomp/ntfaq_18SEP99.html
13. http ://www.clarionmag.com/cmag/news.html ?year=2001&limit=500&
14. http ://www.clarionmag.com/cmag/products.html ?catid=163&subcatid=187
15. http ://root.cern.ch/root/Cint.phtml ?ref
16. http ://www.capesoft.com/docs/multiproj/mptutor.htm
17. http ://www.sparxsystems.com.au/ea_history.htm
18. http ://www.clarionmag.com/cmag/products.html ?catid=163&subcatid=168
19. http ://sparks.discreet.com/knowledgebase/sdkdocs/html/idx_R_class_interface.htm
20. http ://www.ab-archiv.de/Romanian/katalle/u124.htm
21. http ://www.ab-archiv.de/Deutsch/katalle/u124.htm
22. http ://www.softpile.com/authors/Visual_Paradigm.html
23. http ://www.ab-archiv.de/English/kat/u124.htm
24. http ://www.capesoft.com/docs/agent/agent.htm
25. http ://www.clarionmag.com/cmag/products.html ?catid=163&subcatid=188
26. http ://www.capesoft.com/docs/multiproj/multiproj.htm
27. http ://rs1.sze.hu/~tomcat/win32/walk32_1.txt

Classe 3

1. http ://freespace.virgin.net/glynn.etherington/ole.htm
2. http ://www.yukselinan.com/mcpexam/310.txt
3. http ://www.cs.tut.fi/~tensu/dictionary/sanat.html
4. http ://silan_liu.tripod.com/COM.htm
5. http ://progtutorials.tripod.com/COM.htm
6. http ://www.clarionmag.com/cmag/news.html ?year=2002&limit=500&
7. http ://www.hardwareanalysis.com/content/topic/10613
8. http ://lists.dmsolutions.ca/pipermail/mapserver-users/2002-July.txt
9. http ://www.itsteve.com/downloads/allext.txt

10. http ://www.garret.ru/~knizhnik/goods/readme.htm

Classe 4

1. http ://www.capesoft.com/docs/REPLICATE/REPLICATE.HTM
2. http ://info.borland.com/devsupport/bcppbuilder/bugs/bcppbuilder4/bugs/compiler.html
3. http ://www.harmonyware.com/changelog.html

Classe 5

1. http ://www.sitepoint.com/forums/printthread.php ?t=123769&pp=200
2. http ://www.capesoft.com/docs/fm3/fm3.htm

Classe6

1. http ://www.codetools.com/dll/The_DLL_Hell.asp ?df=100&forumid=16684&exp=0&select=676101
2. http ://www.codetools.com/dll/The_DLL_Hell.asp ?df=100&forumid=16684&exp=0&select=675926
3. http ://www.codetools.com/dll/The_DLL_Hell.asp ?df=100&forumid=16684&exp=0&select=675926&fr=26

Recherche d'information sur Internet par algorithmes évolutionnaires

Résumé

Dans ce travail de thèse, nous présentons le problème de recherche d'information sur Internet et plus généralement de veille stratégique. Nous remarquons généralement qu'il est nécessaire de passer beaucoup de temps à analyser les résultats fournis par les moteurs de recherche traditionnels afin d'obtenir une réponse satisfaisante. Dans cette thèse, nous avons donc développé un outil de recherche automatique basé sur une stratégie de recherche évolutionnaire. Cet outil explore les pages Web en partant des résultats fournis par les moteurs de recherche traditionnels (comme Google, Altavista, ...). Plusieurs méthodes d'optimisation ont été comparées : une approche génétique, une approche à base de population de fourmis et un algorithme tabou. L'effort de recherche a également été parallélisé et peut être distribué sur plusieurs machines distantes afin de maximiser les ressources disponibles à l'exécution de cette tâche et d'utiliser une architecture parallèle de grande ampleur. Enfin, nous proposons un système de visualisation des résultats d'un moteur de recherche basé sur les propriétés des nuages d'agents afin d'aider les utilisateurs à mieux comprendre les éléments renvoyés par le moteur et de diminuer ainsi le temps nécessaire à leur analyse.

Mots-clés

moteur de recherche, algorithme génétique, fourmi artificielle, algorithme tabou, classification textuelle, nuages d'agents

---

Searching for information on Internet with evolutionary algorithms

Abstract

In this thesis, we present the problem of information search on Internet and more generally of strategic watch. We generally notice that it is necessary to spend a long time to analyze the results provided by traditional search engines in order to obtain good answers. In this thesis, we have developed an automatic search tool based on evolutionary search strategies. This tool explores Web pages starting from the results provided by traditional search engines (like Google, Altavista ...). Several methods of optimization have been compared : a genetic approach, a population of ants and a tabu search. The search effort is also parallelized and can be distributed on several distant computers in order to maximize the resources available for the execution of this task and in order to user a large scale parallel architecture. Finally, we propose a display system of search engine results based on the property of flock of agents in order to help the users to better understanding the results returned by the engine and to decrease the time necessary to analyse these results.

Keywords

search engine, genetic algorithm, artificial ant, tabu algorithm, textual clustering, flock of agents