**île Rousse 2005**
**Journée sur les systèmes d'information élaborée**

# *EXPERIMENTATIONS DE METHODES PARALLELES DE RECHERCHE D'INFORMATION SUR INTERNET*

**Fabien PICAROUGNE (*), Gilles VENTURINI(*), Christiane GUINOT(**)**
fabien.picarougne@univ-tours.fr , venturini@univ-tours.fr , christiane.guinot@ceries-lab.com

(*) : Laboratoire d'Informatique, Université de Tours, 64, Avenue Jean Portalis, 37200 Tours, France
(**) : CE.R.I.E.S, 20 rue Victor Noir, 92521 Neuilly sur Seine Cedex, France

**Mots clés** : Recherche d'information, Moteur de recherche, Algorithme génétique (AG), Algorithme de fourmis, Méthode tabou, Algorithme parallèle, Internet

**Résumé**
Nous proposons dans cet article 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 et d'optimisation (AG, algorithme à base de fourmis artificielles, 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.
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 à un méta-moteur utilisant notre fonction d'évaluation basée sur la requête de l'utilisateur. 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. Enfin, dès lors que l'utilisateur spécifie une requête avec beaucoup de détails, notre approche génétique fournit majoritairement de meilleures réponses.

# 1. 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. Pour tout utilisateur (entreprise ou autre), 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. Ces outils fournissent souvent en réponse à une requête des milliers de documents en un temps inférieur à la seconde. Cependant, dans bien des cas, l'utilisateur passe un temps assez long à analyser les résultats du moteur, sans garantie de résultats.

Nous proposons ainsi 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 et d'optimisation 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.
Plus précisément, nous allons tester un algorithme génétique, 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.
Nous avons précédemment établi que le problème de recherche d'information sur Internet peut être modélisé comme un problème d'optimisation : Internet est un graphe où les nœuds représentent les pages Web et où les arcs orientés correspondent aux hyperliens entre les documents [1, 2, 3]. Une fonction d'évaluation peut être définie pour chaque point en utilisant la requête de l'utilisateur. Cette fonction numérique quantifie l'adéquation d'une page à cette requête. Nous avons également montré dans [4] comment un AG séquentiel peut être utilisé pour résoudre ce problème et obtenir de meilleurs résultats que les moteurs de recherche classiques. Cependant, le temps nécessaire à l'obtention des résultats est très long. Ce problème peut être résolu en parallélisant la recherche. Cela nécessite l'utilisation d'une méta-heuristique massivement parallèle. Nous avons par conséquent proposé dans [5] une parallélisation de cet AG appelée GeniminerII qui utilise un AG parallèle améliorant à la fois le temps d'exécution et la qualité des résultats obtenus.
Toutefois, il peut être intéressant de confronter cette stratégie d'exploration à d'autre type de méta-heuristique. Nous proposons dans ce papier deux nouvelles stratégies de recherche d'information s'appliquant sur notre modèle inspirées toutes deux de problématiques d'optimisation : une première méthode basée sur un algorithme de fourmis ainsi qu'une deuxième s'inspirant d'une méthode tabou.

La suite de cet article est organisée comme suit : dans la section 2 nous présentons le modèle utilisé pour transformer le problème original en un problème d'optimisation. Dans la section 3 nous détaillons le fonctionnement de nos trois méta-heuristiques en rappelant les principes de GeniminerII. La section 4 est consacrée aux études expérimentales et à la comparaison entre les trois méta-heuristiques et un méta-moteur de recherche. Finalement, la section 5 présente les perspectives qui peuvent être dérivées de ce travail.

# 2. Un modèle pour la recherche d'information

## 2.1. Approches précédentes

Plusieurs applications des AG pour la résolution de problèmes relatifs à Internet peuvent être mentionnés [6, 7, 8, 9]. Cependant, il n'y a principalement que le travail suivant [10] qui est dévolu à l'amélioration du parallélisme de la recherche. InfoSpider est un AG qui manipule une population d'agents. Ces agents sont sélectionnés selon la pertinence des documents retournés à l'utilisateur. L'AG optimise les paramètres de recherche des agents selon les sélections effectuées par l'utilisateur.
Dans notre approche, nous modélisons le problème à un niveau proche d'un paysage de qualité ("fitness landscape") : l'AG n'optimise pas les paramètres des agents de recherche, mais traite directement avec les points de l'espace de recherche (voir la section suivante). De plus, la requête dans InfoSpider est relativement simple. La fonction d'évaluation est interactive : cela signifie que l'utilisateur aide l'algorithme à toujours trouver de meilleurs documents. Dans notre cas, nous sommes intéressés par une approche automatique et donc complémentaire. Cependant, on peut noter que les deux approches ne sont pas incompatibles : au lieu d'utiliser de simples agents, InfoSpider peut utiliser des copies d'agents génétiques basés sur notre modèle. Nous pouvons également incorporer dans notre algorithme une évaluation interactive où l'utilisateur peut souligner les pages intéressantes. Ces pages peuvent alors être utilisés pour affiner la fonction d'évaluation (voir la section 3.1 sur la fonction d'évaluation qui prend en considération la similarité entre les pages Web cherchées et une ou plusieurs pages cibles).
Il y a un nombre important d'algorithmes à base de colonies de fourmis créés afin de résoudre des problèmes de réseau. Par exemple, [11, 12] essayent d'explorer efficacement le Web dans le but de trouver les meilleurs chemins dans le réseau pour résoudre des problèmes de routage. D'autres applications essayent de construire des réseaux optimisés [13] ou de détecter des intrusions dans un réseau [14]. Des algorithmes de classification basés sur des approches à bases de fourmis artificielles ont aussi été appliqués au web [15, 16] dans le but de catégoriser les utilisateurs de sites web. Mais il n'existe pas à notre connaissance d'application spécifique des algorithmes de fourmis à la recherche d'information sur Internet.
La recherche tabou est une méthode d'optimisation mathématique de la famille des techniques de recherche locale présentée par Fred Glover en 1986 [17]. 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 [18]. Mais également, il n'existe pas à notre connaissance d'application spécifique des algorithmes tabou à la recherche d'information sur Internet.

## 2.2. Le modèle

Comme mentionné dans l'introduction, ce problème peut être formulé comme un problème d'optimisation. Internet est un espace de recherche qui contient beaucoup de points (les pages Web), et qui est structuré comme un graphe avec des relations de voisinages (les hyperliens). Les points peuvent être évalués par une fonction d'évaluation qui calcule la qualité d'une page en fonction de la requête de l'utilisateur. Des opérateurs de recherche peuvent être définis : la création d'adresse IP au hasard (mais qui mène à de faible résultats [19]), la création heuristique de pages qui sont obtenus par l'interrogation des moteurs de recherche standards (de manière similaire à un méta-moteur), et l'exploration locale des liens dans les pages Web.
L'exploration locale des liens peut être considérée comme un opérateur d'ascension locale ("hill climbing") ou comme un opérateur de mutation. L'utilisation d'une méta-heuristique pour ce problème mène à deux principales améliorations : 1) la requête de l'utilisateur peut être encore plus précise si on considère que la page entière est analysée en détail, 2) une stratégie efficace peut être définie (en décidant où affecter l'effort de recherche, voir la section 3). À partir du moment où tous les éléments requis par les méta-heuristiques sont présents, on peut définir un AG, un algorithme à base de fourmis artificielles ou encore un algorithme tabou pour la recherche sur Internet.

## 2.3. Fonction d'évaluation et opérateurs de recherche

Nous décrivons dans un premier temps l'évaluation et les opérateurs employés dans les méta-heuristiques. Le tableau 1 montre tous les critères que nous avons actuellement définis dans notre fonction d'évaluation. L'utilisateur doit donner un ensemble de mots clés ($K_1$, $K_2$, ...). Nous définissons plusieurs catégories de mots clés, qui respectivement doivent/ne doivent pas ou devraient/ne devraient pas être présents dans les pages analysées (respectivement ($M_1$, $M_2$, …) / ($M_{n1}$, $M_{n2}$, …) et ($S_1$, $S_2$, …) / ($S_{n1}$, $S_{n2}$, …)). Le contenu des documents retournés peut être spécifié précisément avec les critères $C_{F1}$ à $C_{Sim}$ : les fichiers qui doivent être contenus dans la page Web, ou la similarité de la page avec une page existante qui est déjà connue pour être intéressante. Nous affectons ensuite un poids pour chaque critère. Tous les poids sont uniformes par défaut mais l'utilisateur peut les augmenter ou les diminuer. Pour comparer deux pages $P_1$ et $P_2$, nous calculons chaque critère un par un. Le score de la page $P_1$, par exemple, est la somme des poids de chaque critère qui sont tels que $P_1$ est supérieur à $P_2$. De cette façon, nous évitons tous les problèmes dus aux facteurs d'échelle dans les nombreux critères.

**Tableau 1.** *Les différents critères utilisés dans la fonction d'évaluation des pages.*

| Critères | Définition |
|---|---|
| $C_{K1}$ | # de mots clés ($K_1$, $K_2$, ...) présents |
| $C_{K2}$ | favorise les pages dans lesquelles la proportion de mots du texte correspondant aux mots-clés est importante |
| $C_{K3}$ | spécifie que tous les (K1,K2, ...) mots-clés doivent être présents |
| $C_{K4}$ | rapidité d'apparition des mots clés |
| $C_{K5}$ | égale proportion des mots clés |
| $C_{K6...8}$ | favoriser les mots clés en gras (resp. italiques, soulignés) |
| $C_{K9}$ | favoriser la proximité de couples de mots clés sélectionnés |
| $C_M$ | tous les ($M_1$, $M_2$, ...) mots clés doivent être présents |
| $C_{Mn}$ | aucun des ($M_{n1}$, $M_{n2}$, ...) ne doivent être présents |
| $C_S$ | maximiser la présence des mots clés ($S_1$, $S_2$, ...) |
| $C_{Sn}$ | minimiser la présence des mots clés ($S_{n1}$, $S_{n2}$, ...) |
| $C_{F1…F4}$ | nombre de fichiers images (resp. films, sons, PS, PDF, etc) |
| $C_{Size}$ | taille du document |
| $C_T$ | type de page |
| $C_{Sim}$ | maximiser la similarité à une page donnée |

Deux opérateurs sont utilisés pour parcourir l'espace de recherche. L'opérateur de création heuristique $O_{creat}$ utilise les mots clés ($K_1$, $K_2$, ...) donnés par l'utilisateur afin d'interroger les moteurs de recherche classiques et obtenir de nouvelles pages Web. Nous utilisons actuellement les moteurs de recherche Google, Altavista, Teoma, Lycos et Yahoo. Cet opérateur est utilisé afin de récupérer des points de bonne qualité pour les populations initiales de l'AG, de l'algorithme à base de fourmis et pour les points d'initialisation de l'algorithme tabou. Il peut aussi être utilisé durant la recherche (voir section 3.1 pour l'approche génétique, la section 3.2 pour l'algorithme à base de fourmis artificielles et la section 3.3 pour l'algorithme tabou).
L'autre opérateur est un opérateur d'exploration $O_{explo}(P , Dist)$. Il considère une liste de liens présents dans une page Web et possède deux paramètres. Si $Dist = 1$, alors seulement les liens hypertexte présents dans la page $P$ sont pris en considération. Si $Dist = 2$, alors seulement les liens hypertexte présents dans le voisinage direct de $P$ sont considérés, et ainsi de suite pour les autres valeurs de $Dist$. Ce type d'opérateur est largement utilisé en optimisation : il peut correspondre, par exemple, à l'opérateur de mutation dans les AG, et à une exploration locale dans les recherches en profondeur, etc. Quand cet opérateur ne peut arriver à la profondeur désirée (à cause de pages n'ayant pas de liens hypertexte), il stoppe à la plus grande profondeur atteinte. La manière de sélectionner les liens à explorer en premier est un point important de cet opérateur. Pour réaliser cette opération, nous utilisons une heuristique simple : les liens d'une page $P$ sont triés selon la présence des mots clés ($K_1$, $K_2$, ...) dans leur voisinage. Nous considérons ici le texte présent sur l'ancre du lien mais également le

texte immédiatement situé autour du lien. L'opérateur peut alors sélectionner un lien de manière aléatoire, ou seulement le premier lien, ou un parmi les meilleurs liens, etc.
Dans l'AG et l'algorithme tabou, la valeur du paramètre *Dist* est fixée à 1. Dans le cas de l'AG, on peut rapprocher cet opérateur de l'opérateur de mutation dans le sens où il génère de nouveaux individus à l'aide d'autres individus de la population.

# 3. Trois stratégies de recherche

## 3.1. Une approche par algorithme génétique parallèle : GeniminerII

L'algorithme principal de GeniminerII est décrit dans la figure 1. Il utilise les principes suivants : chaque individu dans la population est une page Web. Initialement, la population est vide et elle se remplit progressivement jusqu'à ce qu'elle atteigne une taille maximale de $Pop_{max}$ individus. Les pages filles sont générées de la manière suivante : avec une probabilité de 1 - $P_{mut}$, ou si la population contient un individu ou moins, nous interrogeons les moteurs de recherche standards afin d'obtenir une page. Autrement, nous utilisons l'opérateur de mutation ($O_{explo}$ dans les notations définies à la section précédente) avec une probabilité $P_{mut}$. Afin de déterminer quel individu parent $P \in Pop$ nous allons muter, nous utilisons la sélection par tournoi binaire sur un couple d'individus sélectionnés au hasard (le meilleur des deux individus est gardé pour la mutation). Si tous les liens de la page sélectionnée ont déjà été explorés, alors nous utilisons l'opérateur précédent pour engendrer un descendant. Ensuite, la page fille $P_O$ est téléchargée, analysée et insérée (si possible) dans la population.
Cet algorithme combine les avantages d'une requête riche et filtrante avec la stratégie de recherche génétique. Il bénéficie de l'optimalité des AGs en ce qui concerne la résolution du dilemme d'exploration versus exploitation [20] : il décide quelles pages explorer en priorité et quelles pages éliminer (avant que tous leurs liens ne soient évalués). Il répartit ainsi de manière optimale un nombre d'essais (exploration de liens) aux meilleurs pages observées. Si $P_{mut} = 0$ et si l'utilisateur demande une requête avec uniquement des mots clés simples (i.e. ($K_1$, $K_2$, ...) et aucun autre critère), alors cet algorithme se comporte exactement comme un méta-moteur de recherche. Au contraire, plus $P_{mut}$ augmente, plus l'algorithme va explorer les liens trouvés dans les pages à défaut des liens donnés par les moteurs.

**Figure 1.** *Algorithme principal de GeniminerII.*

1) **Définir** la fonction d'évaluation f en fonction de la requête de l'utilisateur,
2) $Pop_G \leftarrow \varnothing$;
3) **Faire en parallèle**
4) **Générer** une page $P_O$ (un descendant) :
    a) avec une probabilité 1 - $P_{mut}$ (ou si $|Pop_G| < 2$) alors $P_O \leftarrow O_{creat}$ (page obtenue des moteurs de recherche standards)
    b) Ou avec une probabilité $P_{mut}$ : sélectionner une page fille $P \in Pop_G$ par tournoi binaire et faire $P_O \leftarrow O_{explo}(P,1)$ (exploration de liens de P) ;
    Si P n'a pas de liens à explorer Alors **Enlever** P de $Pop_G$
5) **Insérer** $P_O$ dans $Pop_G$ si ($P_O$ a des liens non explorés) et ($|Pop_G| < Pop_{Gmax}$ ou $f(P_O) > \min_{p \in PopG}\{f(p)\}$),
6) **Aller à** 3 ou **Stopper**,
7) **Fin du parallélisme**,
8) Retourner la meilleure page.

Plusieurs modèles d'AGs parallèles ont étés proposés dans la littérature depuis le début du développement des AG [21, 22, 23, 24]. Dans notre problème, le point crucial est de paralléliser l'évaluation et le téléchargement des pages : à cause des latences des réseaux, le temps perdu durant le transfert des pages entre le serveur et nos machines est très important au regard du temps nécessaire à la comparaison de plusieurs individus et à la manipulation de la population. Nous avons par

conséquent décidé d'utiliser une population centralisée dans notre algorithme génétique en utilisant des opérations de lecture/écriture asynchrones.
Nous utilisons k copies de l'AG (voir la boucle de parallélisation dans la figure 1) qui peuvent effectuer les opérations de téléchargement/évaluation. Lorsqu'un nouvel individu est généré par une copie de l'AG, il l'insère dans la population centralisée. À cette étape, un sémaphore protège l'accès à la population. Cette opération est néanmoins très rapide et ne pénalise pas l'ensemble de l'architecture tout en préservant l'intégrité de la population. Cette méthode de parallélisation est très intéressante lorsque le temps utilisé pour évaluer un individu est très important comparé au temps nécessaire à exécuter un opérateur, ce qui est le cas ici.

### 3.2. Une approche par algorithme de fourmis parallèle : Antsearch

L'algorithme Antsearch que nous proposons est une adaptation de l'algorithme API [25] dans le contexte de la recherche d'information dur le Web. API s'inspire du comportement de fourragement des fourmis de la famille *Pachycondyla apicalis* qui peut être modélisé comme un problème d'optimisation : 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 $A_{local}$ (en général $A_{local} < A_{site}$) 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 mémorisés 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 $A_{site}$. API a aussi été modifié de manière à pouvoir simuler plusieurs nids en parallèle, et ce, indépendamment les uns des autres.

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 2.3.
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 $O_{creat}$ (qui interroge les moteurs de recherche classiques), ou selon une probabilité $1\text{-}P_{creat}$ 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.

La plupart des algorithmes à base de fourmis 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 avec une implémentation réellement concurrente (codé en C++ sur un système Windows en utilisant des threads). Deux niveaux de parallélisation sont présents. Premièrement, chaque nid et sa population de fourmis sont simulés en parallèle. Dans chaque nid, les fourmis sont simulées séquentiellement. Le second type de parallélisation se situe à un plus bas niveau de l'architecture logicielle : le téléchargement des pages est réalisé en parallèle avec un maximum de 10 téléchargements simultanés. Nous ne détaillerons pas ces deux niveaux de parallélisation du fait de leur complexité : des sémaphores sont utilisés afin d'éviter de télécharger plusieurs fois les même pages, les pages téléchargées sont stockés au niveau du nid afin de minimiser l'utilisation de la bande passante, etc.

### 3.3. Une approche par algorithme tabou : TabuSearch

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. 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.
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. La transformation de la méthode tabou dans notre problématique consiste à parcourir le Web de page en page à la recherche de documents pertinents en suivant les liens hypertextes les reliant.

**Figure 2.** *Algorithme principal de TabuSearch.*

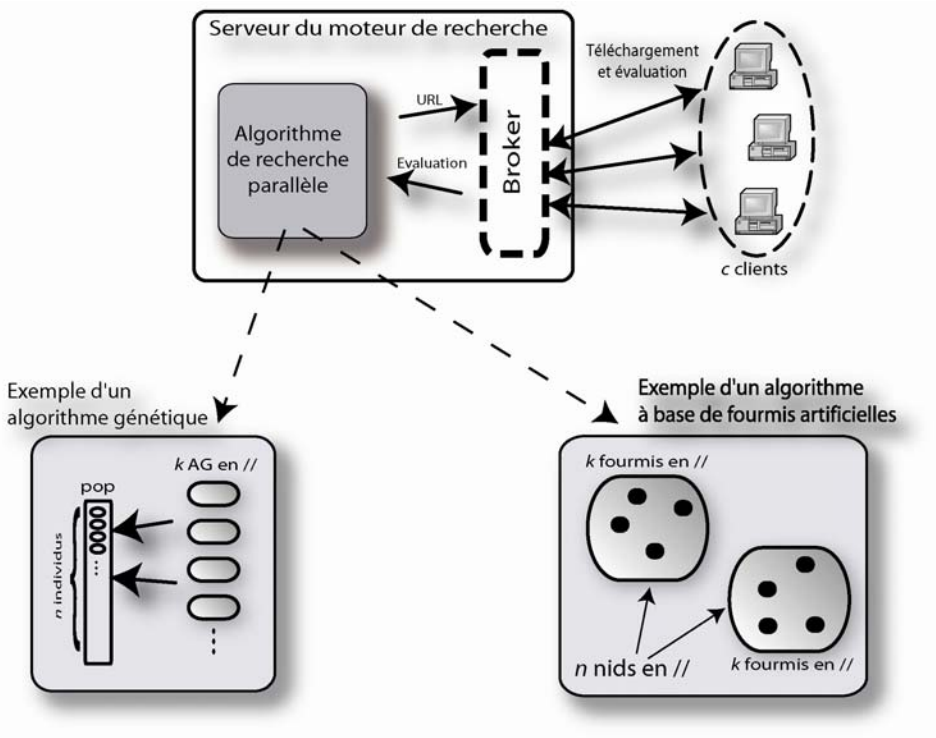
1) **Définir** la fonction d'évaluation f en fonction de la requête de l'utilisateur,
2) *StopAlgo* ← faux
3) $P \leftarrow O_{creat}$
4) **Ajouter** $P$ à la liste tabou
5) **tant que** (*StopAlgo*=faux) et la condition d'arrêt n'est pas vérifiée **faire**
    a) **répéter** $P' \leftarrow O_{explo}(P,1)$
    b) **jusqu'à** ($P' \notin$ liste tabou) ou (il n'existe pas de voisin non tabou de $P$)
    c) **si** il n'existe pas de voisin non tabou de $P$ **alors**
        i) **si** il n'existe pas de pages issues de moteurs de recherche **alors** *StopAlgo* ← vrai
        ii) **sinon** $P \leftarrow O_{creat}$
    d) **fin si**
    e) **si** la liste tabou est pleine **alors**
        Remplacer le dernier élément de la liste tabou par la solution $P'$
    f) **sinon** Ajouter $P'$ à la liste tabou
    g) $P \leftarrow P'$
6) **fin tant que**

La figure 2 décrit l'algorithme 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 $O_{creat}$ 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 l'algorithme GeniminerII.
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 issue 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 $O_{creat}$. D'autres solutions ont été envisagées mais nous ont conduits 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 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.

Un des problèmes que nous pouvons rencontrer en utilisant un tel algorithme de recherche tabou est de ne pas utiliser suffisamment l'opérateur $O_{creat}$ afin d'explorer de nouveaux points de l'espace de recherche. En effet, le graphe représentant Internet est très fortement interconnecté. Le suivi de liens hypertextes ne mène donc 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 $O_{creat}$ 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.

**Figure 3.** *Architecture distribuée du moteur de recherche*



### 3.4. Distribution de la recherche

Nous avons construit notre modèle en pensant à une architecture massivement distribuée. Elle est basée sur une architecture client/serveur qui permet de distribuer l'effort de téléchargement et d'évaluation des pages Web sur plusieurs clients. Ceci nous permet d'utiliser tout algorithme qui peut identifier des opérations maître/esclave dans notre modèle distribué.
Nous considérons ici *c* clients (machines ou processeurs ou threads) qui ont été enregistrés sur le serveur et sont disponibles pour la recherche parallèle. Le modèle que nous avons adopté consiste en 1) centraliser l'algorithme principal sur le serveur, 2) utiliser plusieurs threads effectuant des requêtes de téléchargement/évaluation en parallèles, 3) envoyer toutes les opérations de téléchargement/ évaluation à un *broker* qui distribue l'effort de recherche sur les *c* clients. Un client reçoit l'URL des pages à évaluer. Il télécharge les pages et les évaluent. Lorsqu'une page est téléchargée et évaluée, le client retourne l'évaluation au thread originaire de la requête (voir figure 3). On peut noter également que chaque client peut télécharger plus d'une page à un moment donné. Ce point est très important pour exploiter le temps perdu par la latence du réseau.
A partir du moment où l'algorithme de recherche utilisé est capable d'effectuer plusieurs opérations de téléchargement/évaluation en parallèle, cette architecture permet d'exploiter un calcul distribué sans changement dans l'algorithme, de manière transparente. Nous souhaitons à moyen terme autoriser plusieurs clients à se connecter via le Web et créer ainsi une architecture massivement parallèle.

## 4. Résultats

### 4.1. Méthodologie expérimentale

Dans un premier temps, nous voulons comparer quatre méthodes de recherche d'information sur Internet : GeniminerII, Antsearch, Tabousearch et un méta-moteur de recherche standard. Le méta-moteur utilisé consiste à compiler les résultats donnés par plusieurs moteurs de recherche interrogés par les mots clés ($K_1$, $K_2$, ...) de la requête de l'utilisateur. Afin de comparer les trois approches, nous utilisons une méthodologie proche de celle utilisée en optimisation : pour la fonction d'évaluation donnée, et pour un nombre donné d'évaluations (1000 par exemple), quelle méthode produit les meilleurs résultats et combien de temps requière-t-elle ?

**Tableau 2.** Les 10 requêtes utilisées dans nos tests.

| 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 | | $C_{K3}$ $C_{K4}$ $C_{F7}$ $C_{F8}$ $C_{K9}(2, 3)$ |
| 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 | | | $C_{F1}$ $C_{F2}$ $C_{K3}$ $C_{K9}(2, 3)$ |
| 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 | | | $C_{K3}$ $C_{K4}$ $C_{K5}$ $C_{K9}(1, 2\|4, 2\|3, 4)$ |

Chaque méthode effectue 1000 évaluations de pages Web. Le résultat d'une méthode est un ensemble de pages Web. Pour comparer ces résultats entre eux, nous considérons les nr premières pages trouvées par chaque méthode pour une requête donnée et une exécution donnée. Nous fusionnons ensemble les pages trouvées par les trois méthodes (soit $3 \times n_r$ pages) et les trions selon la fonction d'évaluation. Nous affectons alors à chaque résultat un score $sc_p$ qui est inversement proportionnel au rang des pages Web qu'il contient. Nous totalisons ensuite les scores obtenus par chaque méthode, et nous normalisons les sommes 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 n'est pas sensible à la différence de grandeur des critères utilisés pour évaluer les pages. Nous avons sélectionné 10 requêtes qui sont présentées dans le tableau 2.

## 4.2. GeniminerII vs Antsearch vs Tabusearch

Nous avons résumé les résultats obtenus dans les tableaux 3, 4 et 5. Pour chaque requête, nous avons comparé les résultats obtenus par les trois méthodes réglées avec un paramétrage optimal testé expérimentalement sur un grand nombre d'exécutions pour plusieurs requêtes. Nous avons ainsi pu déterminer le paramétrage suivant pour chaque algorithme :

- Pour GeniminerII, $Pop_{Gmax}$ = 100 et $P_{mut}$ = 0.6, ce qui garantit à la fois une bonne couverture de l'espace de recherche et une bonne exploitation des individus présents dans la population
- Pour Ansearch il est préférable d'utiliser une faible amplitude d'exploration pour l'opérateur $O_{explo}$ ($A_{local}$=1 et $A_{site}$=2) et de simuler un petit nombre de fourmis dans un grand nombre de nids afin de couvrir au mieux l'espace de recherche.
- Pour Tabusearch, les expérimentations ont montré qu'il est préférable de simuler un grand nombre de listes tabou de taille maximale relativement importante (40 éléments).

La comparaison entre ces méthodes est réalisée sur les $n_r$=30, 60, 100 premiers résultats obtenus et les résultats respectifs sont indiqués dans les tableaux 3, 4 et 5. 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, etc.

**Tableau 3.** Évaluation comparée sur 10 requêtes des 30 premiers résultats obtenus par GeniminerII, Antsearch, Tabusearch et par un méta-moteur. Un "►" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

| N° | 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] |

Les résultats obtenus en analysant les 30 premiers documents (voir tableau 3) 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.

**Tableau 4.** Évaluation comparée sur 10 requêtes des 60 premiers résultats obtenus par GeniminerII, Antsearch, Tabusearch et par un méta-moteur. Un "►" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

| N° | 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] |

Le tableau 4 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 3. 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.

**Tableau 5.** Évaluation comparée sur 10 requêtes des 100 premiers résultats obtenus par GeniminerII, Antsearch, Tabusearch et par un méta-moteur. Un "►" signale le meilleur résultat obtenu par requête et les écarts types sont donnés entre crochets.

| N° | 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] |

Enfin le tableau 5 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 $O_{creat}$.

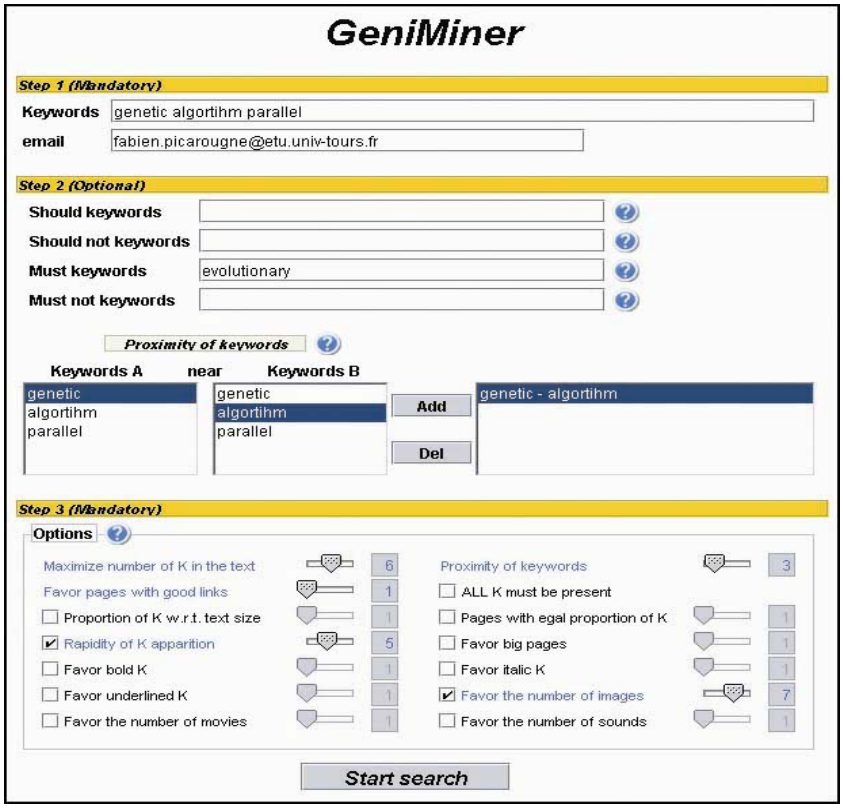
### 4.3. Etude comparative entre l'approche génétique et un méta-moteur de recherche par des utilisateurs réels

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. 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 $O_{creat}$ (voir section 2.3) et à afficher alternativement les résultats obtenus par chaque moteur dans leur ordre d'apparition.

La figure 4 représente une interface graphique réalisée en Java qui permet à l'utilisateur d'interroger GenimMinerII en spécifiant sa requête. Les différentes listes de mots clés définies dans la section 2.3 sont représentées dans le haut de l'interface et l'utilisateur peut également spécifier quels couples de mots clés doivent être favorisés (en ce qui concerne la proximité des mots composants le couple). Chaque critère de la fonction d'évaluation peut être validé dans le bas de l'interface et un poids peut être affecté à chaque critère en utilisant un slider.

**Figure 4.** *Interface graphique 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 réalisés) 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. Les résultats obtenus sur 50 requêtes différentes sont présentés dans les tableaux 6 et 7. Pour chaque tableau, nous avons dissocié les requêtes utilisant uniquement les options par défaut 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 6 présente le nombre de requêtes pour lesquelles une méthode de recherche particulière obtient un meilleur score. Alors que le tableau 7 indique le score cumulé obtenu par chaque méthode de recherche pour toutes les requêtes prises en compte.

**Tableau 6.** *Évaluation par des utilisateurs réels de la pertinence des résultats retournés par GeniMinerII en comparaison à des moteurs de recherche standard.*

| **Vote majoritaire** | ***GeniMinerII*** | ***Moteurs de recherche standard*** | ***Egalité*** |
|---|---|---|---|
| **Toutes les requêtes** | *22 (44,00 %)* | *20 (40.00 %)* | *8 (16.00 %)* |
| **Requêtes filtrées** | *13 (48.15 %)* | *9 (33.33 %)* | *5 (18.52 %)* |

**Tableau 7.** *Évaluation par des utilisateurs réels de la qualité des résultats retournés par GeniMinerII en comparaison à des moteurs de recherche standard.*

| | ***GeniMinerII*** | ***Moteurs de recherche standards*** |
|---|---|---|
| **Toutes les requêtes** | *284 (49.65 %)* | *288 (50.35 %)* |
| **Requêtes filtrées** | *164 (50.62 %)* | *160 (49.38 %)* |

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. 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 6 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 équivalentes (colonne Egalité), 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 du tiers 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.
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.

## 5. Conclusion

Nous avons présenté dans ce papier un nouveau modèle de moteur de recherche sur Internet et trois méta-heuristiques (AG, algorithme à base de fourmis artificielles et algorithme tabou) guidant la stratégie de recherche de documents dans ce réseau. Ces méthodes ont la possibilité d'utiliser des requêtes riches et d'implémenter des stratégies de recherche efficace en comparaison par exemple avec une méta-recherche. La méthode génétique a montré sa supériorité dans le traitement de ce problème. Notre modèle permet de plus de distribuer l'effort de recherche sur des clients distants diminuant ainsi fortement le temps nécessaire à l'exécution de nos requêtes. Ainsi nous argumentons que notre outil est bien adapté à aider un expert humain à réduire le temps passé à analyser les résultats d'un moteur de recherche. Nous préparons actuellement une évaluation plus importante de nos algorithmes en collaboration avec des utilisateurs réels qui vont comparer les résultats donnés par notre outil de recherche à ceux donnés par d'autres moteurs et méta-moteurs de recherche.

Nous sommes également en train d'ajouter au moteur de recherche lui-même d'autres fonctionnalités importantes qui ne sont pas en relation directe avec les AGs : un système de suggestion efficace, les résultats de la recherche vont très prochainement être présentés sous forme hiérarchique en utilisant un algorithme de classification sous forme d'arbre, un système de filtrage collaboratif va aider les membres d'une société à souligner les documents pertinents et un algorithme de construction automatique de site portail peut être utilisé sur les résultats obtenus afin de construire une base de donnée collective sur des sujets donnés. Une application réelle est en cours de développement en collaboration avec le CE.R.I.E.S.

# 6. Bibliographie

[1] ALBERT R., JEONG H., BARABASI A.-L., «Diameter of the World Wide Web», *Nature*, vol. 401, 1999, p. 130–131.

[2] BRODER A., KUMAR R.,MAGHOUL F., RAGHAVAN P., RAJAGOPALAN S., STATA R., TOMKINS A., WIENER J., «Graph structure in the Web», *Proceedings of the Ninth International World Wide Web Conference*, Elsevier, 2000.

[3] PICAROUGNE F., MONMARCHÉ N., OLIVER A., VENTURINI G., «Web mining with a genetic algorithm», *Eleventh International World Wide Web Conference*, Honolulu, Hawaii, 7-11 May 2002.

[4] PICAROUGNE F., MONMARCHÉ N., OLIVER A., VENTURINI G., «Geniminer: Web Mining with a Genetic Based Algorithm», *Proceedings of the IADIS International Conference WWW/Internet*, Lisbon, Portugal, November 13-15 2002, p. 263–270.

[5] PICAROUGNE F., VENTURINI G., et GUINOT C., « Un algorithme génétique parallèle pour la veille stratégique sur internet», *Proceedings of the VSST 2004 conference*, vol. 2, p.519-528, Toulouse, France, 25-29 octobre 2004.

[6] SHETH B. D., «A Learning Approach to Personalized Information Filtering», Master's thesis, MIT Media Lab, January 1994.

[7] MOUKAS A., «Amalthea : information discovery and filtering using a multiagent evolving ecosystem», *Proceedings of the Conference on Practical Applications of Agents and Multiagent Technology*, vol. 11, London, April 1997, p. 437–457.

[8] LAWRENCE S., GILES C.L.: *Accessibility of information on the web*. Nature 400 (1999) 107–109.

[9] VAKALI A., MANOLOPOULOS Y., «Caching objects from heterogeneous information sources», *Proceedings International Database Conference (IDC'99)*, Hong-Kong, July 1999.

[10] MENCZER F., «Complementing Search Engines with Online Web Mining Agents», *Decision Support Systems*, vol. 35, no 2, 2003, p. 195–212.

[11] WAGNER I.A., LINDENBAUM M., BRUCKSTEIN A.M.: *«ANTS: Agents on networks, trees, and subgraphs»*. Future Generation Computer Systems 16 (2000) 915–926.

[12] YANOVSKI V., WAGNER I., BRUCKSTEIN A.: *«*A distributed ant algorithm for efficiently patrollinga network. Algorithmica 37 (2003) 165–186.

[13] WILKINSON S., WHITE T.: *The antsynnet algorithm: «Network synthesis using ant colony optimization»*. In: Proceedings of the 2004 International Conference on Artificial Intelligence (IC-AI '04), Las Vegas (2004).

[14] CARO G.D., DORIGO M.: Antnet: *«Distributed stigmergetic control for communications net-works»*. Journal of Artificial Intelligence Research (JAIR) 9 (1998) 317–365.

[15] ABRAHAM A., RAMOS V.: *«Web usage mining using artificial ant colony clustering and genetic programming»*. In Sarker, R., Reynolds, R., Abbass, H., Tan, K.C., McKay, B., Essam, D., Gedeon, T., eds.: Proc. of the 2003 Congress on Evolutionary Computation CEC2003, Canberra, IEEE Press (2003) 1384–1391.

[16] TELES W.M., WEIGANG L., RALHA C.G.: *«Antweb - the adaptive web server based on the ants' behavior»*. In press, I., ed.: Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03), Halifax, Canada, Morgan Kaufman (2003) 558–561.

[17] GLOVER F.. *«Future paths for integer programming and links to artificial intelligence»*. Comput. Oper. Res., 13(5) :533–549, 1986.

[18] GLOVER F. et LAGUNA F. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[19] LAWRENCE S., GILES C.L.: *Accessibility of information on the web*. Nature 400 (1999) 107–109.

[20] HOLLAND J. H., *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.

[21] CANTÚ-PAZ E., *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers, 2000.

[22] MUHLENBEIN H., «*Evolution in Time and Space – The Parallel Genetic Algorithm*», 1991.

[23] SPIESSENS P., MANDERICK B., «A massively parallel genetic algorithm: Implementation and first analysis», BELEW R., BOOKER L., Eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991, Morgan Kaufman, p. 279–286.

[24] WHITLEY D., STARKWEATHER T., «Genitor II : A distributed genetic algorithm», *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 2, 1990, p. 189–214.

[25] MONMARCHE N., VENTURINI, G., SLIMANE, M.: On how Pachycondyla apicalis ants suggest a new search algorithm. Future Generation Computer Systems 16 (2000) 937–946