Un algorithme génétique parallèle pour la veille stratégique 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 clefs :

Veille stratégique, recherche d'information, moteur de recherche, algorithme génétique, algorithme parallèle, Internet

Keywords:

Strategic watch, information retrieval, search engine, genetic algorithm, parallel algorithm, internet

Palabras clave :

Vigilancia estratégica, recuperación de datos, motor de búsqueda, algoritmo genético, algoritmo paralelo, internet

Résumé

Nous présentons dans cet article un algorithme génétique (AG) parallèle qui explore le Web dans le but de trouver des documents pertinents dans le contexte de la veille stratégique. Nous montrons comment le problème de recherche d'information sur Internet peut être modélisé en un problème d'optimisation : Internet est un espace de recherche structuré sous forme de graphe, et une fonction d'évaluation peut être définie à partir de la requête de l'utilisateur. L'AG gère une population de pages Web et décide quelles pages explorer. L'architecture parallèle de GeniMinerII est distribuée sur un réseau local ou sur Internet, chaque client ayant la possibilité de s'enregistrer et de devenir une partie du moteur de recherche. Nous montrons également que l'AG parallèle obtient de meilleurs résultats en comparaison avec un méta moteur de recherche, et qu'il diminue fortement le temps nécessaire à l'obtention des documents. Enfin, les premiers tests réalisés avec des utilisateurs réels montrent le potentiel de ce système et les orientations futures à prendre en compte.

1 Introduction

La veille stratégique est une activité cruciale dans les entreprises qui est utilisée dès qu'une décision importante doit être prise, par exemple en ce qui concerne la concurrence. En dépit de leur capacité à répondre en quelques secondes, les moteurs de recherche standards à base d'index ne sont pas adaptés à ce problème : la requête de l'utilisateur est très simple et ne filtre pas assez efficacement les résultats. En pratique, une activité de veille stratégique utilisant les moteurs standard implique bien souvent l'obtention d'une réponse en quelques secondes suivie par des heures d'analyse des résultats. Diminuer la part du temps "humain" dans cette activité est un thème de recherche crucial. Nous apportons dans cet article une contribution qui consiste en une meilleure méthode de formulation de la requête associée à une stratégie de recherche efficace dans le but d'automatiser la recherche stratégique sur Internet.

Récemment, nous avons 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, 13]. 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 [12] comment un AG séquentiel peut être utilisé pour résoudre ce problème et obtenir de meilleurs résultats que les moteurs de recherche classiques. Cependant, l'architecture séquentielle de cet AG impose une contrainte lourde pour un moteur de recherche. Nous considérons que dans le contexte de la veille stratégique, tous les ordinateurs d'une entreprise donnée peuvent être considérés comme une architecture parallèle pour exécuter la recherche. L'AG *steady state* (i.e. ne générant qu'un unique individu à chaque itération) utilisé dans le cas précédent génère les individus un par un et passe beaucoup de temps, pendant l'étape d'évaluation, à attendre la réponse des serveurs Web. En conséquence, 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 proposons dans cet article une nouvelle approche appelée GeniMinerII qui utilise un AG parallèle améliorant à la fois le temps d'exécution et la qualité des résultats obtenus.

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 présentons à la fois l'état de l'art et l'AG parallèle proposé. La section 4 est consacrée aux études expérimentales et à la comparaison entre l'AG séquentiel, GeniMinerII 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 [14, 9, 10, 4, 8, 16]. Cependant, il n'y a principalement que le travail suivant [7] 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).

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 [6]), 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 3.2). À partir du moment où tous les éléments requis par les métas heuristiques sont présents, on peut définir un AG pour la recherche sur Internet.

3 Un algorithme génétique parallèle

3.1 Fonction d'évaluation et opérateurs de recherche

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}	prop. aux (K_1, K_2, ...) mots clés en fonction de la taille du texte
C_{K3}	prop. aux (K_1, K_2, ...) mots clés en fonction des autres
C_{K4}	tous les (K_1, K_2, ...) mots clés doivent être présents
C_{K5}	rapidité d'apparition des mots clés
C_{K6}	égale proportion des mots clés
$C_{K7...9}$	favoriser les mots clés en gras (resp. italiques, soulignés)
C_{K10}	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

Nous décrivons dans un premier temps l'évaluation et les opérateurs génétiques. 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 contenu 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.

Deux opérateurs sont utilisés pour parcourir l'espace de recherche. L'opérateur de création heuristique 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 la population initiale de l'AG. Il peut aussi être utilisé durant la recherche par l'AG (voir la section 3.2). L'autre opérateur est un opérateur de mutation. Il considère une liste de liens présents dans une page Web. Il sélectionne aléatoirement un des meilleurs liens de la page (au sens de notre évaluation) et retourne cette nouvelle page en tant que descendant.

3.2 Architecture distribuée de l'AG

GeniMinerII 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 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'individu sélectionné 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 (voir la discussion sur la parallélisation dans la section suivante).

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 [5] : 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.

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 $P_O \leftarrow$ *page obtenue des moteur de recherche standard*
    b) Ou avec une probabilité P_{mut} : sélectionner une page fille $P \in Pop_G$ par tournoi binaire et faire $P_O \leftarrow$ *Mutation* (P) (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.

Figure 1. *Algorithme principal de GeniMinerII.*

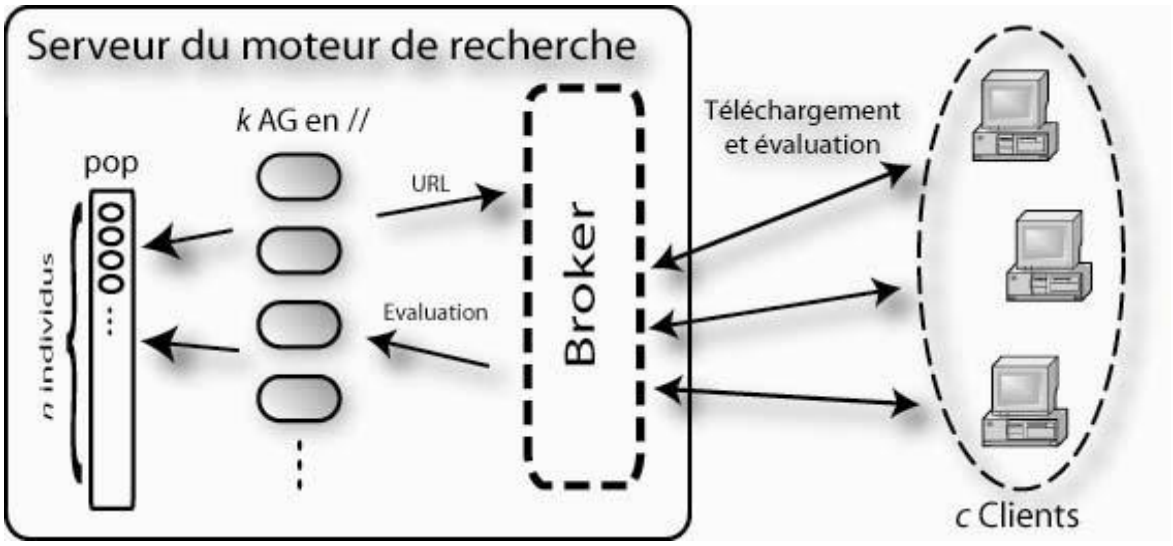

Figure 2. *Serveur du moteur de recherche.*

Plusieurs modèles d'AGs parallèles ont étés proposés dans la littérature depuis au moins le début du développement des AG [3, 11, 15, 17]. Nous considérons ici que c clients (ordinateurs, processeurs ou threads) se sont enregistrés sur le serveur et sont disponibles pour la recherche parallèle. Le modèle que nous avons adopté ici consiste à 1) centraliser la population sur le serveur avec des opérations d'entrée/sortie asynchrones, 2) utiliser k copies de l'AG (voir la boucle de parallélisation dans la figure 1), 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 la page et l'évalue. Le client retourne ensuite l'évaluation à la copie de l'AG correspondante (voir la figure 2). Ensuite, la copie de l'AG insère l'individu dans la population centralisée sur le serveur. 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. On peut également noter que chaque client peut télécharger plus d'une page en même temps. Ce point est important car il permet de réduire le temps perdu par la latence du réseau.

4 Résultats

4.1 Interface utilisateur

La figure 3 représente une interface graphique réalisée en Java qui permet à l'utilisateur d'interroger GeniMinerII. Les différentes listes de mots clés définies dans la section 3.1 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.

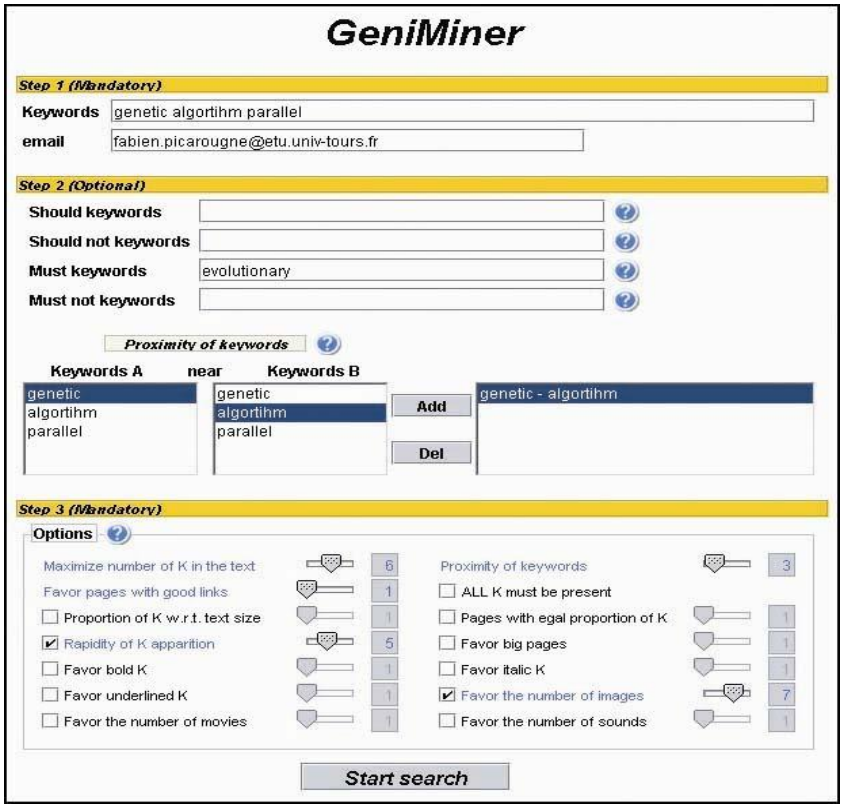

Figure 3. *Interface graphique sous forme d'applet Java.*

4.2 Méthodologie expérimentale

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_{K4} $C_{K10}(4, 5)$ C_{F3}
2	ant algorithm genetic	termit			C_{K1} C_{K4} C_{K5}
3	technology fiber optic information	network	crisis		C_{K4} C_{K5} C_{F7} C_{F8} $C_{K10}(2, 3)$
4	javascript window opener	tutorial free		code	C_{K2} C_{K4} C_{K8} C_{Size}
5	mouse disney movie animation	DVD			C_{K1} C_{K4} C_{K5} C_{F2}
6	text poet flower wind	Baudelaire	Rimbaud		C_{K2} C_{K5} C_{K7} C_{K9} C_{F3}
7	tv reality show	internationnal			C_{K4} C_{F1} C_{F2} C_{K3} $C_{K10}(2, 3)$
8	dll export class template	code example			C_{K1} C_{K4} $C_{K10}(1, 2\|3, 4)$
9	wine excellent price good	Bourgueil	Bordeaux		C_{K2} C_{K5} C_{K7}
10	genetic algorithm artificial ant	experimental comparison			C_{K3} C_{K5} C_{K6} $C_{K10}(1, 2\|4, 2\|3, 4)$

Dans un premier temps, nous voulons comparer trois méthodes de recherche d'information sur Internet : GeniMiner, GeniMinerII 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 3. *Résultats obtenus sur les 10 requêtes.*

Requête		**Méta**	**GeniMiner**		**GeniMinerII**	
N°	**# 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

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. Il n'est pas sensible à la différence de grandeur des critères utilisés pour évaluer les pages. Nous avons sélectionné 10 requêtes qui sont présentés dans le tableau 2.

Nous avons résumé les résultats obtenus dans le tableau 3. Pour chaque requête, nous avons comparés 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} = 0.5 et P_{mut} = 0.3). La taille de la population génétique des deux méthodes est limitée à 100 individus. Cette valeur a été déterminée expérimentalement comme celle apportant les meilleurs résultats.

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, etc.

Nous pouvons immédiatement remarquer dans le tableau 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 standard à base d'index.

Les temps mis par le méta moteur de recherche parallèle et par 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 la qualité des résultats obtenus par GeniMiner. Pour 7 requêtes, l'algorithme parallèle obtient de meilleurs résultats que l'algorithme séquentiel.

4.3 Temps d'exécution

Nous avons testé toutes les méthodes sur un PC standard (Pentium 4 à 2.4 GHz, 512 Mo de RAM) avec une parallélisation basée sur les threads. Comme on peut le constater dans le tableau 4, augmenter le nombre de threads d'exécution fait décroître le temps d'exécution de l'algorithme. Ce temps n'est pas linéaire parce que le temps requis pour télécharger une page est variable. Donc, avec un petit nombre de thread d'exécution, nous avons une plus grande probabilité d'avoir tous les threads en train de télécharger une page Internet lente et d'avoir de ce fait une relativement lente exécution de notre algorithme.

Tableau 4. *Temps d'exécution en fonction du nombre de thread d'exécution pour 1000 téléchargements (test réalisés sur la requête 8).*

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

4.4 Évaluation 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. La requête est spécifiée grâce à l'interface graphique présentée dans la section 4.1. 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 de création de l'AG (voir section 3.1) et à afficher alternativement les résultats obtenus par chaque moteur dans leur ordre d'apparition.

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 premiers résultats obtenus sur 24 requêtes différentes sont présentés dans les tableaux 5 et 6. Pour chaque tableau, nous avons dissociés 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 5 présente le nombre de requêtes pour lesquelles une méthode de recherche particulière obtient un meilleur score. Alors que le tableau 6 indique le score cumulé obtenu par chaque méthode de recherche pour toutes les requêtes prises en compte.

Tableau 5. *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	10 *(41,67 %)*	9 *(37,50 %)*	5 *(20,83 %)*
Requêtes filtrées	6 *(35,29 %)*	6 *(35,29 %)*	5 *(29,41 %)*

Tableau 6. *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	142 *(50,00 %)*	142 *(50,00 %)*
Requêtes filtrées	99 *(50,77 %)*	96 *(49,23 %)*

Même si le nombre de requêtes prisent en compte ne permet pas de généraliser, certaines observations peuvent être faites. GeniMinerII obtient une évaluation similaire voire légèrement supérieure à celle obtenue par les moteurs de recherche classiques. Notre méthode peut donc apporter des améliorations à la recherche produite par les outils classiques de recherche.

Sur les requêtes pour lesquelles les utilisateurs ont considérés les méthodes équivalentes (colonne Egalité), l'évaluation obtenue est majoritairement assez faible (pour 4 requêtes, elle est inférieure à 5). 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 fréquences 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 modèle pour la veille stratégique et un AG parallèle qui a été intégré à un moteur de recherche. GeniMinerII à la possibilité d'utiliser des requêtes riches et d'implémenter une stratégie de recherche efficace comparé par exemple à une méta recherche. De plus, la parallélisation de la recherche de GeniMinerII n'a pas fait décroître les performances en terme de précision, comparé à GeniMiner, et a très largement diminué le temps de calcul. 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 sont 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 gens d'une compagnie 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] CANTÚ-PAZ E., *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers, 2000.

[4] FAN W., GORDON M. D., PATHAK P., «Automatic generation of matching functions by genetic programming for effective information retrieval», *Proceedings of the 1999 Americas Conference on Information Systems*, Milwaukee, WI, USA, August 13-15 1999, p. 49–51.

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

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

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

[8] MONMARCHÉ N., NOCENT G., SLIMANE M., VENTURINI G., «Imagine : a tool for generating HTML style sheets with an interactive genetic algorithm based on genes frequencies», *IEEE International Conference on Systems, Man, and Cybernetics (SMC'99)*, vol. 3, Tokyo, Japan, October 12-15 1999, p. 640–645.

[9] MORGAN J., KILGOUR A., «Personalising information retrieval using evolutionary modelling», *Proceedings of PolyModel 16 : Applications of Artificial Intelligence*, 1996, p. 142–149.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

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