國立中山大學資訊工程學系

碩士論文

運用基因規劃於網際網路搜尋

Web Search Using Genetic Programming

指導教授：李 宗 南 教授
研 究 生：吳 建 興 撰

中 華 民 國 九十一 年 七 月

學年度：91
學期：2
校院：國立中山大學
系所：資訊工程所
論文名稱(中)：運用基因規劃於網際網路搜尋
論文名稱(英)：Web Search Using Genetic Programming
學位識別：碩士
語文別：eng
學號：8934624
提要開放使用：是
頁數：37 頁
研究生(中)姓：吳
研究生(中)名：建興
研究生(英)姓：Wu
研究生(英)名：Jain-Shing
指導教授(中)姓名：李
指導教授(英)姓名：Cl
關鍵字(中)：網
關鍵字(中)：基
關鍵字(中)：網
關鍵字(中)：資
關鍵字(英)：In
關鍵字(英)：G
關鍵字(英)：W
關鍵字(英)：Information Retrieval

中文摘要

從網際網路上定位及檢索所需要的資訊是一重要的問題。目前所存在的搜尋引擎給予過多無用且重複的資訊。由於搜尋的架構隨著搜尋引擎的不同而不同，對於所有的主題，要找到一個合適的搜尋機制是非常困難的。在本論文中，我們提出了一套基因規劃網際網路搜尋系統，可以根據使用者的興趣而自動產生查詢句。此系統可以從搜尋引擎上檢索資訊，並將搜尋結果過濾中去掉重複性與無用的資訊，然後結果將以單一的使用者介面呈現給使用者。而實驗的結果使用本系統所得的查詢結果比起由隨機來產生查詢句所得的查詢結果，與使用者興趣的相似度獲得改善。

Abstract

To locate and to retrieve the needed information from the Internet is an important issue. Existing search engines may give too much useless and redundancy information. Due to the search feature is different for different search engines, it’s very difficult to find an optimal search scheme for all subjects. In this paper, we propose a genetic programming web search system (GPWS) to generate exact query according to a user’s interests. The system can retrieve the information from the search engines, filter the retrieved results and remove the redundancy and useless results. The filtered results are displayed on a uniform user interface. Compared with the queries generated by randomly, the degree of similarity of results and user’s interests are improved.

Contents

List of Figures

List of Tables

1. Introduction

In recent years, with the growing use of the Internet, lots kinds of websites have been set up. However, it is extremely difficult to retrieve needed information among the overwhelming information on the Internet, hence databases and search engine are used. One can retrieval the needed information on the Internet, and then reorganize, manage information and knowledge via a search engine.

Most web search engines response a query from a user, then retrieve the relevant URLs from their own database, which records various kinds of the URLs. However, most of search engines can not give the best index for all subjects. As a result, it gives too much unnecessary information. Due to the search feature is different for different search engines, it's very difficult to find an optimal search scheme for all subjects [1].

Various kinds of approaches in Web Search have been proposed in the last decade. Aggarwal [2] proposed a WWW-based information retrieval and extraction (WIRE) system. This system can perform general search, special search (for example, search telephone number in Web pages), filter information, and extract information. Oard [3] proposed a hybrid method to merge Natural Language Processing (NLP) and Information Retrieval (IR) methods, including the latent semantic indexing method and neural networks. Since the search engines based on artificial intelligence (AI) gain popular, there are more researches in web search using an agent based system [4-8]. Jain [9] proposed a simplified analytical model to evaluate the mobile agent and traditional client-server technology for retrieving information. In his experiment, the mobile agents system has a better performance than the traditional client-server technology. Murugesan [10] introduced some intelligent agents for retrieving

information on the Internet and Web.

However, the existing systems don't focus on user's interest. For example, if one wants to search Web pages about classical music, and he may enter query "classical music", then search engine retrieves the URLs about 'classical music', 'classical' and 'music'. But if query changes as "classical music + Beethoven", then search engine retrieves URLs more relevant to 'classical music' than the "Beethoven". A better query can make the search easier and make the retrieved result closer to what a user needs. When a query is closer to user's interest, the result has fewer parts to be adjusted, hence it saves time and resources.

Evolution Algorithm (EA) can yield an optimal solution for most problems. It can be used to generate a better query. A better query can make the search easier and make the retrieved result closer to what a user needs, it may be good idea to use EA to generate queries. Nick [1] proposed a method that uses Genetic Algorithm (GA) in generating a good query, but the method has some problems. First, it generates too many queries for a user to evaluate all the searched results. Second, it generates query in a fixed format, but it is difficult to generate queries within a fixed format when a user is interested in a variety of areas.

Genetic Programming (GP), which is proposed by Koza [11], is also one kind of evolution algorithms. Because GP can get better performance in handling Boolean operation and queries often are the combination of keywords and Boolean operators, hence, GP is more suitable for performing web search than GA. It uses parsing trees constructed from a function set and a terminal set as individuals, and selects the individual based on fitness to perform crossover process and reproduction process.

Based on genetic programming, we propose a genetic programming web search system (GPWS) to generate exact query according to user's interest, remove the redundancy results, filter results and show results in a uniform user interface. The GP algorithm is used to learn user's interests. Since users' interests are not always the same, they may change from time to time. Hence GPWS keeps observing users' navigation on the Internet and recording users' interests. GPWS is one kind of multi-agents systems. It focuses on the query rather than result, therefore the core of system is to generate query.

The remainder of this thesis is organized as follows. Chapter 2 gives background materials and reviews current approaches. The description of GPWS framework is given in Chapter 3. Experimental results are presented in Chapter 4. Conclusions are given in the last Chapter.

2. Background Materials and Literature Reviews

In this chapter, background material and literature reviews are described. Since information retrieval (IR) is the way of finding information from lots of documents, all search engines use some methods in IR as core technology. GPWS uses information retrieval and the genetic programming as two key technologies. Hence, we described these technologies in Chapter 2.1. After discussing background materials, the current approaches for web search are given in Chapter 2.2.

2.1 Background Materials

There are two important technologies for GPWS: information retrieval and genetic programming. Since, the information requirements are not always precisely

known, it is useful to be able to specify a list of terms that give a good indication of which result are relevant. There are several methods for fast searching and ranking results in IR and they become kernel technology of search engines. genetic programming is the kernel for generating query of GPWS, therefore it is also described in background materials.

2.1.1 Information Retrieval Technology

Since the volume of homepages is huge, how to find the required homepages in a short time becomes a problem for every search engine. There are several methods in IR to retrieve and rank useful information from massive data. Li [12] proposed a marsha chinese question answering system. The evaluation of this system is done by using a method based on the TREC question-answering track. Lawrie [13] defined summarization in terms of a probabilistic language model. He used the definition to explore a new technique for automatic generating topic hierarchies by applying a graphtheoretic algorithm, which is an approximation of the dominating set problem (DSP). Many search engines use these methods as core technology. The main mechanism in IR is to analyze the data, and to rank the data according to the degree of similarity to queries.

In IR, there are several functions to verify the relevance of data to query. One of these methods is vector space [14]. The vector space refers to query and document, and the degree of similarity of a page to a query. It is defined as the cosine value of query and document and is written as

$$\cos(\boldsymbol{Q}, \boldsymbol{D_d}) = \frac{\mathbf{1}}{\boldsymbol{W_q W_d}} \sum_{t \in Q \cap D_d} (\mathbf{1} + \log_e \boldsymbol{f_{d,t}}) \cdot \log_e (\mathbf{1} + \frac{\boldsymbol{N}}{\boldsymbol{f_t}}) \quad (1)$$

where Q is denoted as Query, D_d is the retrieved document; W_q is the weight of a

query; W_d is the length of a document and is used as the weight of page; $f_{d,t}$ is the frequency of the terms of document; f_t is the number of the documents which contain the query term, N is the number of documents, t is one query term in a set of queries.

In order to be more compatible for web search, some parameters in equation (1) are slightly changed here. Since title information and page head can stand for the whole homepage, to compute the degree of similarity of a title and the homepage header is faster than to compute that of whole homepage. Hence, we use title and the homepage header as document instead of using the whole pages. Since the amount of homepages on the Internet is so huge that is difficult to collect these homepages and store them in user's storage just for computing the similarity of Query and founded document, for our propose f_t is redefined as the number of documents which contain the query term among documents which are sent back to the user and N is the total number of documents which are sent back to the user.

2.1.2 Genetic Programming

The genetic programming (GP) is one kind of evolution algorithms. It uses the evolution rules to find the optimal solution just like the creature evolution does. First, it initializes population of individuals, or candidate solutions. Second, it evaluates the population according to the fitness rules. After evaluation, it chooses individual based on the fitness. And then based on the probability, it selects operator to perform crossover or reproduction as the same as the creature. Finally, it evaluates the population size of new generation, and moves the population of new generation to population, then jumps back to evaluate the individuals process until the termination conditions is satisfied.

There are some major differences between GP and GA. GP changes the bitstream individual of GA into program individual. Every individual's fitness in GP is made according to the result of executing the program, but the individual's fitness of GA is made according to the fitness rules. In other words, GP changes the traditional evolution algorithm search space of bitstream into the search space of program behaviors.

2.2 Literature Reviews

There exist numerous literatures on web search. They use several methods in IR as kernel skill from whom it performs different services. In this chapter, we review some literature on web search engines, and then web search using the genetic algorithm.

2.2.1 Web Search Engine

Currently, the most popular way to retrieve the information on the Internet is to use web search engines. Henzinger [15] surveys ranking algorithms used to retrieve information on the Web. Natural language queries improve precision and parsing capability, and with advances in the technology, it can also meet these e-business shopping sites' performance demands [16]. Search engines appear poised to make some major breakthroughs in relevancy ranging and personalization that promise to increase the accuracy and reliability of search. On the other hand, some researches reveal that users are becoming increasingly disenchanted with search engines, because they don't actually search the web, but rather search records of the web sites their robots have visited [17]. Thong [18] presented an audio and video search engine to tackle the problem of the rising need to perform searches on the multimedia contents.

Existing web search engines have some problems. When they retrieve the queries from users, they may refine different results, due to every search engine has its search feature. Sometimes, they return results irrelevant to user-required information. Besides, search engines cooperate with huge databases to make sure that whatever a user wants to find is in database. It's difficult to maintain such a database to weed out the URLs, which is no longer existed, or pointed to an error address, or redundant URLs. When users send query to a search engine, useless URLs may be sent back. Thus they have to check whether these URLs are useful manually.

A general search engine can neither give preference indexing for the homepages, nor classify homepages in very clear way. One feasible solution for a search engine to classify homepages is to promote the owners of the homepages specify class names, when they register a link to a search engine. However, owners may not clearly know what their homepages belong to which classes.

2.2.2 Web Search Using Genetic Algorithm

To retrieve results based on a user's interests, Nick and Thems [1] present a multi-agent system using GA to perform the learning process. It observes users' behavior when they browse the Internet, and then records subject that users usually search, or records users' input keywords, and then store information in users' profiles.

Based on a user's profile, it generates queries to search engine. After obtaining the search result, the useless results are discarded. Then it checks results with user's profile, and send the most relevant ones to users. Finally, it adjusts a user's profile according to the user's feedback to agent.

It uses several agents as illustrated in Figure 1, to perform web search. Among these agents, the GA agent is the most important. The GA agent collects and evaluates new Homepages retrieved from web search engine according to dictionary, which is constructed from the examples provided by users, and historical records of users' behaviors.

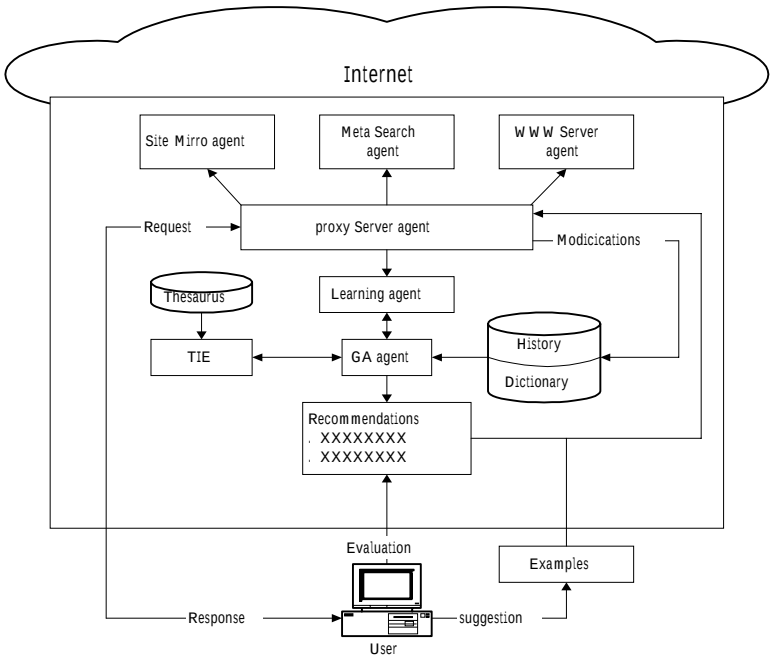

Figure 1. The framework of Web Search Using GA

It used two GAs. One is primary GA to initialize the most useful keywords, and the other is secondary GA to initialize the operators like AND, OR, and NOT. Via these two GAs, a query is produced by the following steps. First, the dictionary, which is considered as user's interests, is constructed according to the example documents given by users. Second, initialize the population of primary GA (PGA) by randomly selecting fixed number of keywords from dictionary. Third, the secondary GA (SGA) randomly selects operators that are correspond to the individual of PGA. Finally, the query is created by combining members from both populations.

The system provides much more relevant query to meet the user's interest, and it

also provides a good way to keep tracking of user's interests when a user's interests are changed. It also performs good recall rate, because the results are produced based on the user's interests. However, the system has some drawbacks with it. First, it uses two GAs to generate the query—with each individual of PGA, the SGA generates a set of operators. In other words, if PGA generates one group of population with size M1, and SGA generates the other group of population with size M_2, then there will be M_1*M_2 of queries. It is difficult for a user to evaluate all the searched result of queries when M_1 and M_2 become large.

Second, it's not able to perform user's interest within fixed format. For example, if a user only wants to read the homepages about "soccer", he has to fill the other five fields with other irrelevant words, and SGA must generate corresponding OR operator or NOT operator; on the other hand, if a user is interested in many things at the same time, using a fixed number of fields may not be able to stand for all his interests.

Third, the dictionary is constructed from examples, which are provided by the user, but it causes a problem when the user is interested in part of the keywords. The irrelevant keywords often occupy most of the example documents. The found keywords of examples are not able to stand for all the user's interests.

3. The Genetic Programming Web Search System Framework

As described above, the existing search engines have some disadvantages. To overcome these disadvantages a new efficient method for performing information retrieval is needed. When combination of keywords and Boolean operators are used to

generate queries for search engine, GP can perform better than GA. Because the GP can perform more efficiently than GA on Boolean operations, in this thesis a Genetic Programming Web Search system (GPWS) is proposed for retrieving Web information.

There are three query models in GPWS: queries are generated by a user; by GPWS; and both of them. The three search models in GPWS are described in Chapter 3.1, and the system architecture is given Chapter 3.2. The GP algorithm, which is the core of GPWS, is presented in Chapter 3.3.

3.1 Query Model

There are three query models in GPWS: manual model, automatic model, and semi-automatic model.

Manual Model

In manual model, a user treats GPWS as a kind of search engine, and sends the query to GPWS. After finish searching, GPWS sends results to the user, and he/she may send feedback to GPWS to refine results. And in this model, GPWS only records his/her query terms, updates the user's profile, and obtains the user's feedback to adjust user's profile.

Automatic Model

In automatic model, a user treats GPWS as a kind of robot, and sends a signal to command GPWS to perform search process. After finish searching, GPWS sends the most relevant results to the user, and he/she may send the feedback to GPWS to refine result. And in this model, GPWS generates query, and obtains the user's feedback to adjust user's profile.

Semi-automatic Model

In semi-automatic model, a user treats GPWS as a kind of search engine, and sends few query terms to GPWS. When GPWS obtains the query terms, it treats these keywords with high priority query terms, and generates the query including these high priority query terms with other keywords in the user's profile. After finishing search, GPWS sends results to the user, and he/she may send the feedback to GPWS in order to get a finer result. And in this model, GPWS records his/her query terms, generates the query, which includes these query terms and other keywords in the user's profile, updates the user's profile, and obtains the user's feedback to adjust user's profile.

3.2 Genetic Programming Web Search Architecture

The GPWS system consists of two subsystems – a Web subsystem and a GP subsystem. Figure 2 shows the GPWS architecture. The Web subsystem handles protocol communication with the Internet. The GP subsystem is consisted of four agents: learning agent, adjusting agent, interface agent, and query generation agent. The GP subsystem handles communication with a user, such as user's queries, a user's profile, and so on.

3.2.1 The Web Subsystem

The jobs of the web subsystem are to communicate with the Internet. When a user wants to access the search engine by himself, one thing he/she doses is just to send a query to GP subsystem and after the GP subsystem records the query terms, the query is sent to the Web subsystem; or a user sends a signal to the GP subsystem, and let the GP subsystem generate the query. Then the web subsystem accesses web search engine like Google, and sends the results back to the GP subsystem.

Since web search engine technologies become mature, GPWS just focuses on that how to generate query and sends query to search engine.

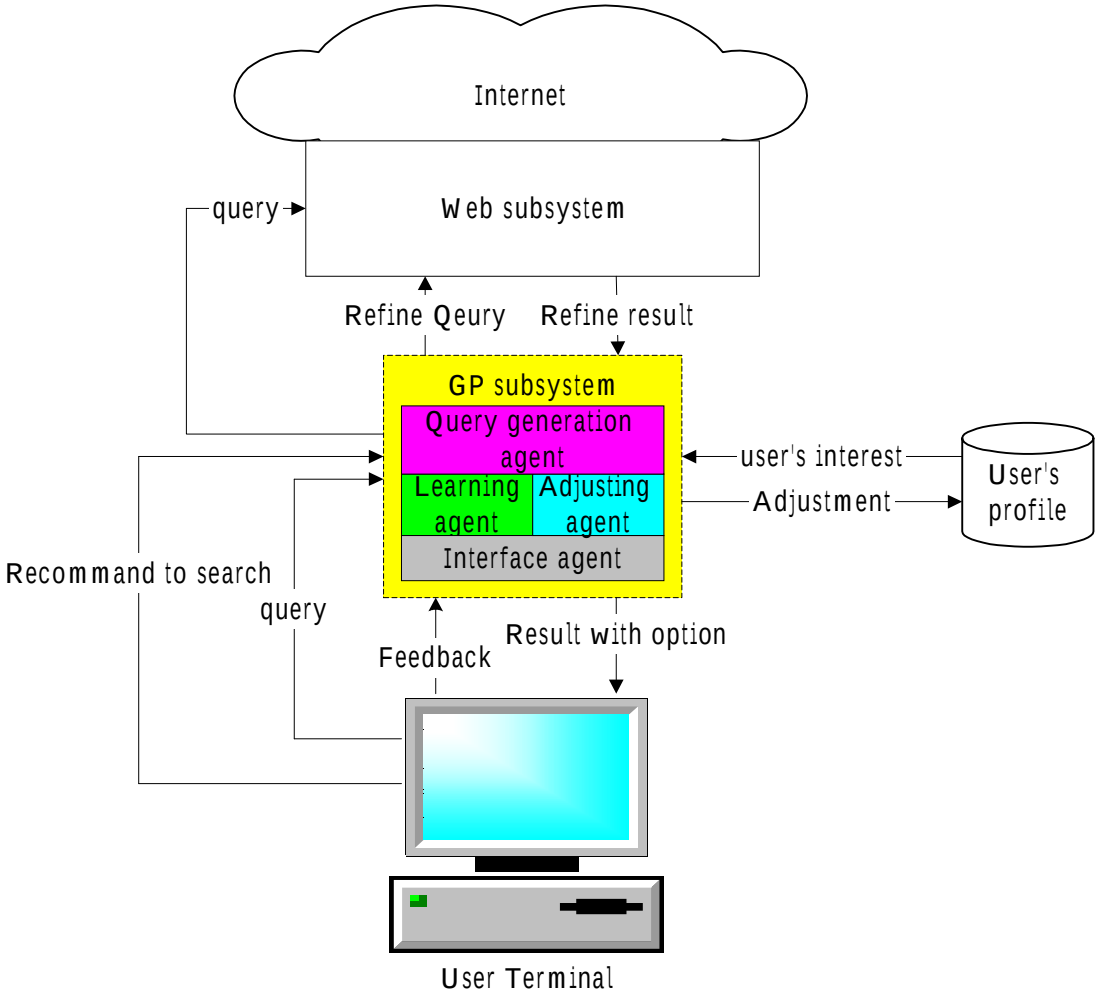

Figure 2. The Framework of GPWS

3.2.2 The Genetic Programming Subsystem

The kernel of GPWS is the GP subsystem. It consists of four agents: learning agent, adjusting agent, interface agent, and query generation agent. Each of them has its jobs and cooperates with others. They share the data and may use results produced by other agents, as its inputs.

As illustrated in Figure 2, the GP subsystem determines the query and

refine-solution according to a user's profile to generate query and send to web subsystem, and get result. After the GP subsystem combines all results collected by the web subsystem, it finds the most relevant results and sends them back to the user interface. The GP subsystem can get feedback from the user to adjust the user's profile and to refine result of pervious execution. Repeat these steps until the user is satisfied with the results. All agents are described as follows.

Learning agent

The jobs of learning subsystem are to learn a user's interest while the user is exploring the Internet and to construct the user's profile. A user can send the query to the web subsystem via the GP subsystem, and the web subsystem sends query to search engine. When the GP subsystem obtains the queries from the user, learning agent records the keywords of these queries. Next the learning agent fills these keywords into the user's profile, and if the keywords are already recorded in user's profile, then it updates the weight of these keywords in the user's profile, otherwise it records the keywords and assigns the weights.

After the search engine finishes its job and sends the result back to Web subsystem, the Web subsystem sends the results to the GP subsystem, and then the learning agent wakes up the adjusting agent to adjust the keywords' weights.

The user's profile is used in query generation agent and adjusting agent of the GP subsystem. The file's content, which is shown in Table 1, records the user's interest keywords, how many documents contain the keyword, the number of times to be searched, and the weight of the keyword.

Table 1:The content of a user's profile

Keywords	Documents	Frequency	Weight
Intelligent	11	24	30
Game	5	16	15
JAVA	5	16	15
…	…	...	…

Adjusting agent

The job of adjusting agent is to adjust keywords' weight using the feedback of search result. Table 2 defines the scale value of adjustment. The role of the adjusting part is to adjust the user's profile and to let the file much closer to a user's interest. When the user changes his interest, the system changes the profile at the same time.

Table 2: The incremental value of adjustment

	Incremental value on query keywords			The terms not in the query
User's feedback	AND	OR	NOT	
Strong Interesting	2	1	-2	2
Interesting	1	0	-1	1
Indifferent	0	0	0	0
Irrelevant	-1	-1	1	-1
Strong Irrelevant	-2	-2	1	-2

Table 2 shows the adjustment of query terms' operators. Notice that the field of "The terms not in the query", it means that the keyword is not in the query that generated by the GP subsystem, but it is shown in the result. For example, one generates the query, which is "game AND sport", and the content of result sometimes overlaps with "PCgame" and "Video game". These two terms are not in the query, but in the results. Hence, the adjusting agent adjusts these two terms' weight in profile.

Interface agent

The job of the interface agent is to display results to a user, and get feedback to the adjusting agent. Since results obtained from different search engines may have different styles, so interface agent must rearrange these results and display them in a uniform format.

There are some problems with the searched results. The searched URLs are sometimes redundant. The titles of these URLs are different, but may link to the same homepage. Besides, the searched URLs sometimes are no longer existed or pointed to an incorrect address. Search engines cannot filter these incorrect and redundancy URLs, hence, sends useless URLs to the user. The interface agent should take care of these problems.

When the interface agent obtains the retrieved URLs, it filters the retrieved URLs by connecting these URLs within a fixed time. If the retrieved URLs haven't give responses, then the interface agent weeds out the useless results. If the retrieved URLs connect to the same address, then the interface agent keeps one useful URL, and weeds out other redundancy URLs.

Query generation agent

The job of query generation agent is to use a user's profile, which is created by learning agent and adjusted by adjusting agent, to generate query similar to the user's interests. The genetic programming algorithm is used in this agent. This algorithm is discussed in the next chapter.

Generating part is not only used to generate new query, but also to evaluate the result. It evaluates all results and sends the most relevant result to interface agent.

When a user wants to refine the search result, he may send more query terms manually or use the GP agent to generate the query terms.

3.3 The Genetic Programming Algorithm

In this chapter, the GP algorithm is described. The definition of GP is given in 3.3.1, and the algorithm is described in 3.3.2.

3.3.1 Definition of Genetic Programming Algorithm

Before describing the algorithms, we gives some definition for GP and fitness rules.

Function set

The function set is denoted as the operating function in genetic programming. function set $\boldsymbol{F}$ for this problem is consisted of three Boolean operators, defined as

$$\boldsymbol{F}=\{\text{AND, OR, NOT}\}$$

Terminal set

The terminal set is denoted as the data of function set $\boldsymbol{F}$ in genetic programming. The terminal set $\boldsymbol{T}$ for this problem consists of keywords in the user's profile denoted as $\boldsymbol{K_i}$, and **i** is denoted as the position in the user's profile. And $\boldsymbol{T}$ is

$$\boldsymbol{T}=\{K_1, K_2, \ldots, K_m \mid \text{where m denotes the number of total keywords}\}$$

Individual

The individual is denoted as one solution in genetic programming. The individual is presented as a parsing tree $\boldsymbol{P_t}$, internal nodes are selected from function set $\boldsymbol{F}$, and the leaf nodes are selected from terminal set $\boldsymbol{T}$. P_t is

$$\boldsymbol{P_t} = \{\text{the parsing tree which is the combination of } \boldsymbol{F} \text{ and } \boldsymbol{T}\}$$

Population

The population is denoted as solution sets in genetic programming. The population $\boldsymbol{P}$ for this problem is consisted of parsing tree $\boldsymbol{P_t}$ defined as

$$\boldsymbol{P}=\{P_t^{(1)}, P_t^{(2)}, \ldots, P_t^{(n)} \mid \text{where n denotes population size}\}$$

First, one operator is chosen from function set F. Second, two or one items are chosen from the union of function set and terminal set according to the first chosen function that needs one argument or two arguments. Figure 3 shows an example of $\boldsymbol{P_t}$.

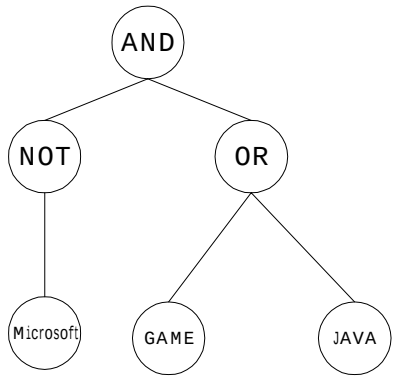

Figure 3. An individual example

Fitness rule

The fitness rule in genetic programming is the objective function. The fitness rule $\boldsymbol{F_R}$ for this problem is defined as the sum of cosine value, of which is the vector spaces of search result and the keywords of query. The higher value of $\boldsymbol{F_R}$ is, the finer result is. $\boldsymbol{F_R}$ is defined as

$$\boldsymbol{F_R} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{N}} \cos(\boldsymbol{Qt_j}, \boldsymbol{D_d}) \tag{2}$$

where *Qt* is denoted as an arbitrary keyword in a user's profile, D_d is the retrieved document and *N* is the total numbers of keywords in a user's profile.

Standard Fitness

The standard fitness in genetic programming is the way to measure the solution. The standard fitness $\boldsymbol{F_S}$ for this problem is defined as the maximum value of $\boldsymbol{F_R}$ minus the current $\boldsymbol{F_R}$, which is

$$\boldsymbol{F_s} = \boldsymbol{F_{R_{Maximum}}} - \boldsymbol{F_{R_{Current}}} \tag{3}$$

3.3.2 The Genetic Programming Algorithm

The GP algorithm used in GPWS consists of initialization process, evaluation process, reproduction process, and crossover process. Detailed steps as illustrated in Figure 4 are described as follows.

GP contains two main parameters -- the ***max_gen*** and ***M***. The parameter ***max_gen*** is the generation of Genetic Programming, and ***M*** is the size of population. There are some minor parameters, like ***Pe*** and ***Pr***. ***Pe*** is the probability to perform the crossover process, and ***Pr*** is the probability to perform the reproduction process. The higher ***Pe*** is, the faster we can get the answer. When ***Pr*** is higher, the finer answer we can get. Due to ***Pe***+***Pr***=1, it is a trade off between two parameters.

The algorithm determines input GP parameters: A set (or population) of candidate solutions ***P***, population size ***M***, max generation ***max_gen***, crossover rate ***Pe*** and reproduction rate ***Pr***, and let ***GEN*** equal zero. First randomize initial individuals of the population. Second, use the individual as query and send to the web subsystem. After the search finished, results are sent to the GP subsystem to evaluate the result. After finishing the evaluation process, it sorts the queries and results based on fitness. Then it sends the most relevant query and results to interface agent of the GP subsystem to display the result in user interface. It sorts the results and queries based on fitness value after evaluation. And then it checks the variable ***GEN***. If the variable ***GEN*** equals the variable ***max_gen*** then returns the relevant results. If ***Gen*** is not equal to ***max_gen***, then it sets variable ***i*** zero. If the variable i equals the variable M, then it adds 1 to variables GEN and goes back to check the variable GEN, else, it generates a random number $K \in [0,1]$. IF $1>K>Pe$, it calls the reproduction process;

else if $Pe > K > 0$ it calls the crossover process. Then it adds 1 to variable to i, and goes back to check the variable i.

Since, the function set only contains three operators – "**AND**", "**OR**", and "**NOT**", but the terminal set contains much more components than the function set, GPWS uses a probability to control the numbers of internodes of parsing tree P_t (functions) to control the tree size of parsing tree P_t. But if the parsing tree becomes too huge, then it is not able to be a query. Thus GPWS checks the height of tree in initialization step. If the height of tree grows over a fixed number, which sets to 4 in GPWS, then the function nodes, which have not chosen the child nodes, must choose the terminal nodes according to its arguments as its child nodes.

GPWS evaluates the search results by computing the degree of similarity between results and the keywords in user's profile. Since the keywords in user's profile are not only collected from the data of a user's examples, but also collected from the user's navigation on the Internet, if a user usually uses the keywords, then these keywords will have higher weights. It uses equation **(2)** and equation **(3)** to calculate the fitness value. The web subsystem retrieves the results, and then it gives the results to the GP subsystem. The evaluation algorithm sets the fitness value of each query to zero and the variable j to 0. And then, it marshals the retrieved results, and filters the keywords. The next step is to compute the cosine value of each document in retrieved results. After computation, it denotes the sum of all cosine value in the retrieved results as fitness value of the query and adds 1 to the variable j. At final, it checks the variable j, if $j < M$ then the execution progress goes back to marshal the retrieved results, otherwise, it returns the fitness value of each queries.

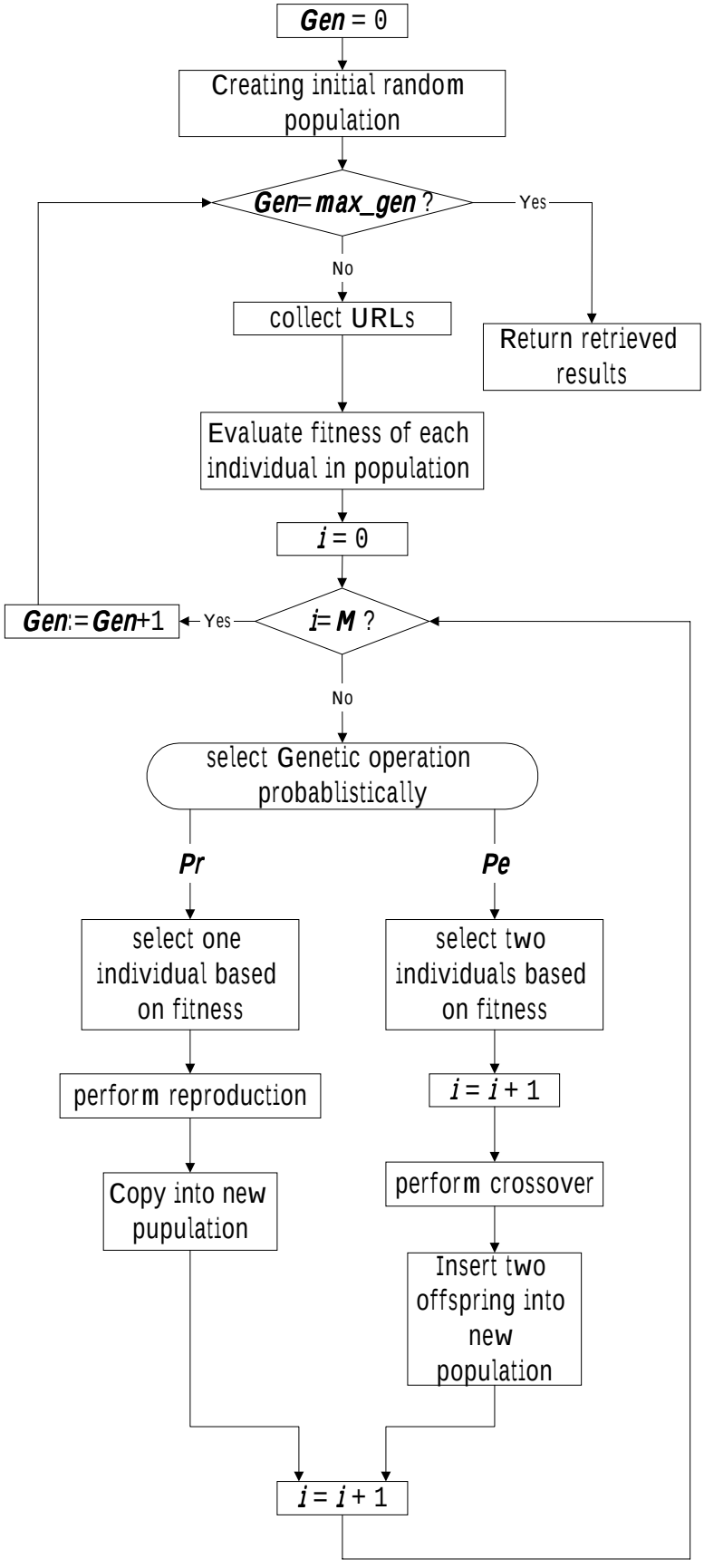

Figure 4. The GP algorithm of GPWS

During the actually using GPWS, the adjusting agent gets the feedback of query and calculates the incremental amount of each keyword of query, which is described in Table 2, and it adjusts the weight of keywords in the user's profile.

The detailed adjusting process steps are described as follows. When the interface agent obtains the user's feedbacks of retrieved results, the algorithm gets all original weights of keywords from user's profile. And then, it computes the incremental amount Δ_i where ***i*** denotes the number of query terms according to the table of the scale value of adjustment. Finally, it computes each adjusted value of each keyword in user's profile and stores the new weights into user's profile.

The only job of reproduction is to reproduce parent individual as new offspring individual in new generation. Reproduction can reserve the fine answer and prevent the answer from becoming worse after performing crossover process several times, so that, the answer can be converged in a short time.

The detailed crossover steps are described as follows. First, the algorithm selects two individual from the population based on fitness and adds 1 to variable ***i***. And then, call the avoid-process, if the return value of avoid-process is true, then it chooses other two individuals based on fitness, otherwise, it performs crossover. Two individuals randomly choose a node of parsing tree without a forbidden crossover mark to be crossover. If there are two parsing tree A and B, both the chosen nodes are root nodes, then inserts the two trees as offspring into the new generation. If one of the chosen nodes are root and the other one isn't, the tree is merged into the other one, and insert only the merged tree as a new offspring into the new generation. If both of the two chosen nodes are not a root, the two nodes and its subtrees change over as

illustrated in Figure 5. It inserts new two offspring into new population finally.

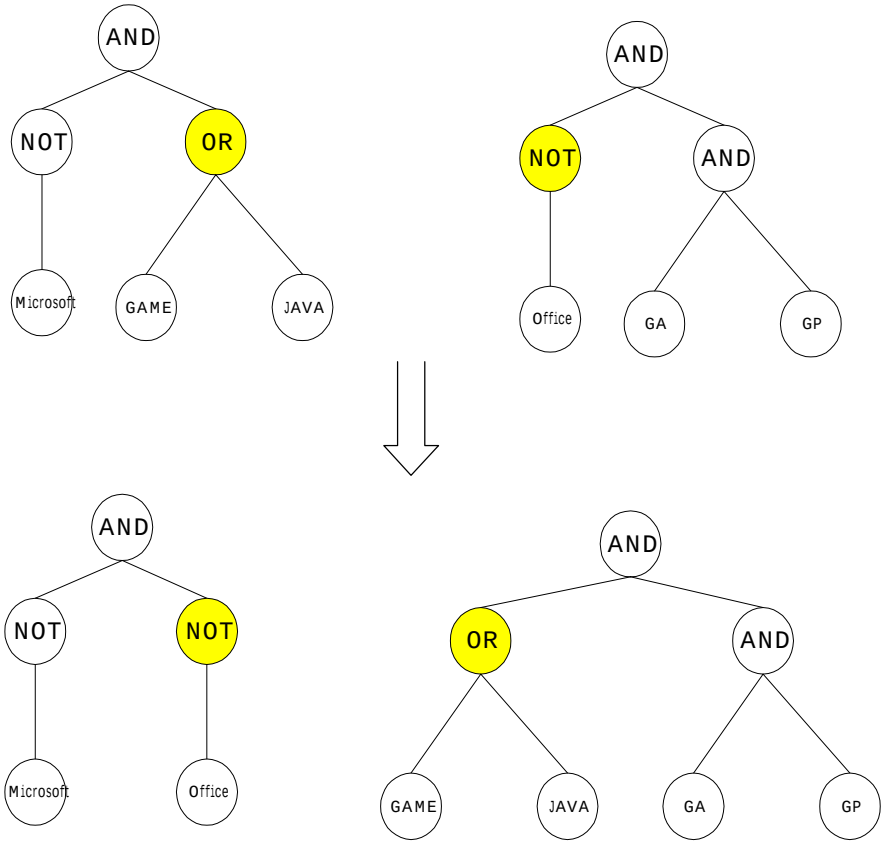

Figure 5. Example of Crossover process

If the same keyword appears twice in one parsing tree, then the whole query may become meaningless, so that keywords in one parsing tree can only appear at most once. For example, query **a** is **- -Linux**, and query **b** is **Linux + JAVA**, after executing crossover process, the new offspring query **A** may become **– JAVA**, and the other new offspring query **B** may become **Linux + -Linux**. The query **B** is meaningless.

One mechanism designed in GPWS is to avoid the duplicate keywords while performing crossover operation. First, the avoid process checks the keywords of two parsing tree. Second, it gives the same keywords a forbidden crossover mark. Third, if a node has a forbidden crossover mark, then it's parent node is also given a forbidden crossover mark.

Crossover process must choose a node without a forbidden crossover mark to perform crossover. If the chosen node has forbidden crossover mark, it has a probability to meet the situation.

Figure 6 shows an example of the avoid operation. When the avoid process examines the keywords of two parsing trees, it finds the term "**GAME**" appears in two trees, then it gives the node "**GAME**" a forbidden crossover mark. After finish checking all keywords, the avoid process examines all nodes of two parsing trees. If the node has a forbidden crossover mark, then all its parent nodes have to give forbidden crossover marks.

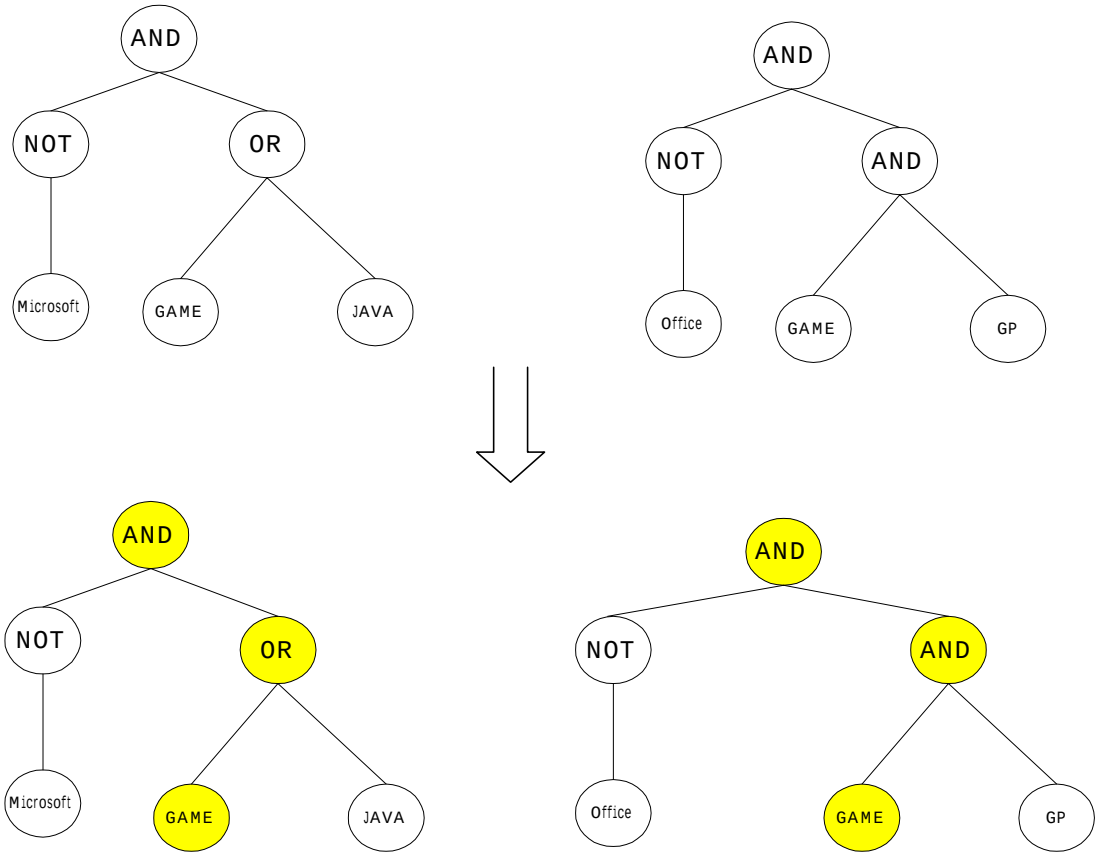

Figure 6.The Avoid-process.

The job of selection is to select one or two individuals from population. It uses the Roulette Wheel method to make the individuals with high weight have more chance to be selected.

4. Experimental Results

In our experiment, a Pentium 4 CPU with 1.5 GHz, 256 MB RAM, 10M bps network speed, JAVA programming is used, and www.yahoo.com is used as the search engine environment. The population size *M* is 20, the number of generation *max_gen* is 10, the crossover rate *Pe* is 0.9, and the reproduction rate *Pr* is 0.1. Table 3 summarizes the key features of web search problem. In first experiment, we use 10 documents from the same area (about programming) as user's examples and collect keywords from the documents. Besides, we insert several keywords to simulate the collected keywords from a user's navigation on the Internet, where its document and frequency columns set to zero and only the weight column set with nonzero number. The user's profile, which is collected from the same area, is illustrated in Table 4. The second experiment, we use 10 documents from different areas (for example, car, motorcycle, comic, programming, music, etc) as the user's examples and collect keywords from the documents. The same as in the first experiment, we insert several keywords to simulate the collected keywords during a user's navigation on the Internet, where its document and frequency columns set to zero and only the weight column set with nonzero number. The user's profile, which is collected from several areas, is illustrated in Table 5.

Table 3: Tableau for the Web search problem

Objective:	Find an appropriate query for a user's interest.
Terminal set	$\boldsymbol{T}$**={**$\boldsymbol{K_1, K_2, \ldots, K_m}$**|where m denotes the number of total keywords}**
Function set	$\boldsymbol{F}$**={AND, OR, NOT}**
Raw fitness	$\boldsymbol{F_R = \sum_{j=1}^{N} \cos(Qt_j, D_d)}$
Standard fitness	$F_s = F_{R_{Maximum}} - F_{R_{Current}}$
Wrapper	None
Parameters	Max generation=*max_gen*, population size = 20
Success predicate	The individuals (query) are satisfying the user.

Table 4: The user’s profile that is collected from the same area

Keywords	Documents	Frequency	Weight
bus	3	7	7
car	3	2	3
cdrom	3	10	9
diablo	3	7	7
doraemon	3	10	9
dream	3	3	5
food	3	6	6
Freebsd	4	15	14
GAME	0	0	10
Gundam	3	10	9
Hacker	4	11	11
Intel	3	10	9
Intelligent	3	24	30
Japen	3	9	8
Java	5	16	15
jbuilder	3	2	3
Linux	5	15	14
motocycle	3	6	6
movie	3	7	7
mp3	3	7	7
music	3	7	7
mysql	4	11	11
network	4	10	10
paper	3	7	7
PHP	4	10	10
Starcraft	4	10	10
story	3	2	3
tcpip	4	11	11
tv	3	9	8
unix	4	13	13
watch	2	2	2
wireless	4	10	10
X-files	0	0	8

Table 5: The user’s profile that is collected from several areas

Keywords	Documents	Frequency	Weight
applet	3	2	2
array	3	2	2
assembly	3	1	1
bean	3	7	7
c	3	10	9
cbuilder	3	3	5
class	0	0	6
database	4	15	14
delphi	5	16	15
double	3	10	9
float	4	11	11
function	3	10	9
html	8	24	30
integer	0	0	8
java	5	16	15
java2	3	2	3
jbuilder	5	15	14
jdbc	3	6	6
masm	3	7	7
mysql	2	7	6
oracle	3	7	7
pascal	4	11	11
php	0	0	10
pointer	3	7	7
program	4	10	10
script	4	10	10
sic	3	2	3
socket	4	11	11
sqlserver	3	9	8
string	4	12	13
swing	2	2	2
vb	4	10	10
xml	3	9	8

In first experiment result, which is illustrated in Figure 7, is the average relevant of each generation. And the content of the simulated user's profile is illustrated in Table 4. We remove the common words (such as "the", "a", "I", etc) and the words with high frequency in 10 documents (such as "like", "love", etc). Since the keywords are from the same area, if one keyword of the user's profile is in the retrieved results, then the probability of that the others keywords of the user's profile appear in the retrieved results is increased. In the experimental result, one can see the degree of similarity of GA is slightly better than GP, but much closer when the generation reaches fifteen. Both GP and GA perform better than the query generated randomly.

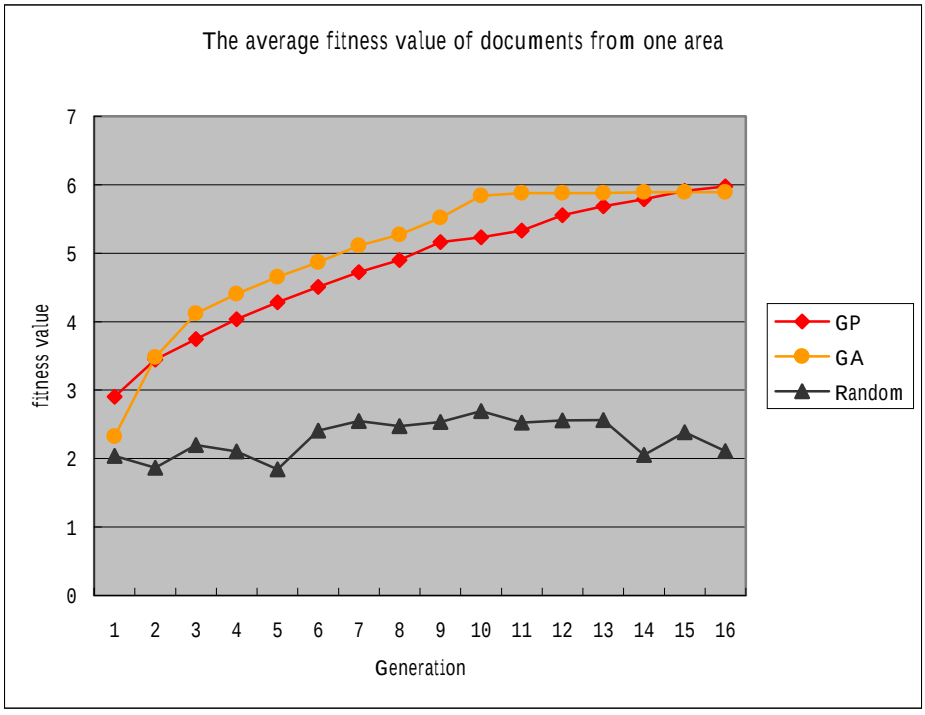

Figure 7. The average fitness value of documents from one area.

Figure 8 shows the average relevant of each generation. The keywords are extracted as the same way with the first experiment. Since the keywords are collected from different area, the keywords in user's profile are not related to each other. In

experimental result, one can see the degree of similarity of GP is better than GA when the generation reaches eight. And both using GP and GA to generate queries based on a user's interests can get higher degree of similarity than to generate queries randomly when the generation greater than three. Even the examples are in different areas, the results show satisfactory performance.

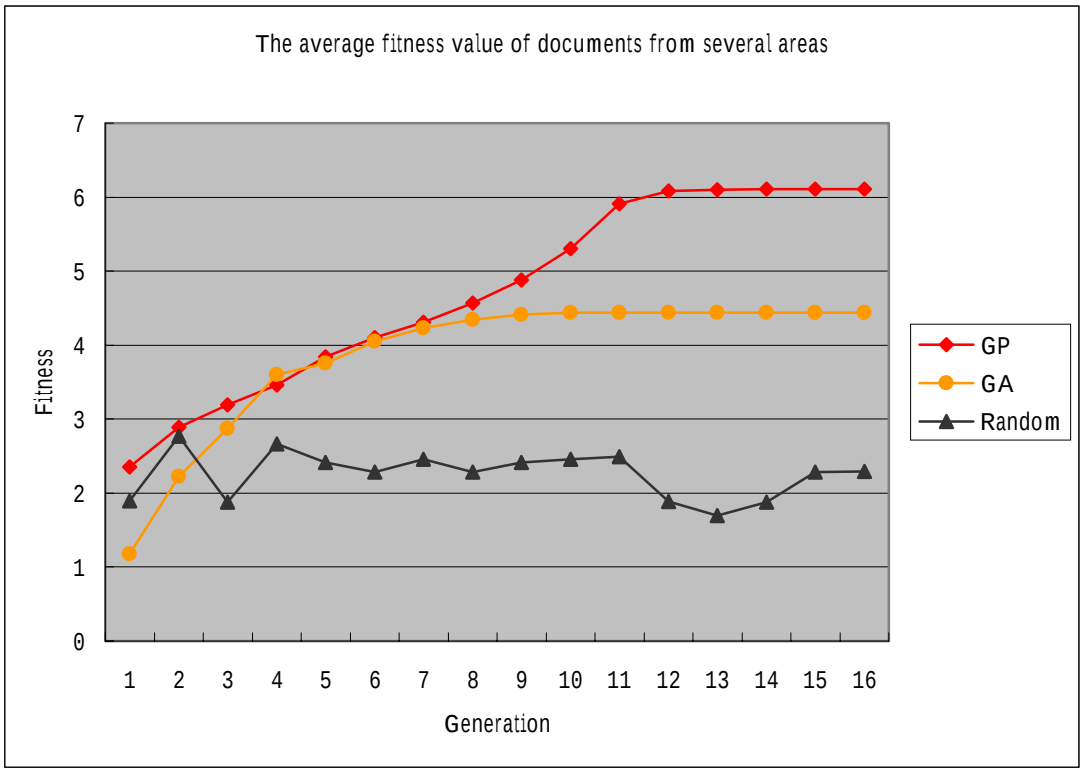

Figure 8. The average fitness value of documents from several areas.

One can see that the degree of similarity of GA performs better than that of GP in the first experiment, but the degree of similarity of GP performs better than that of GA in the second experiment. The reason of difference between the two experimental results is the individual. Because the degree of similarity is computed with the user' profile, the more keywords are retrieved, the higher degree of similarity is. In the first experiment, although the individuals of GP have dynamic keywords, the effect of GP is not significant, because the retrieved keywords are too similar to the keywords in

the user's profile. In the second experiment, the individuals of GP have dynamic keywords, and the individuals can contain more number of keywords than individuals of GA with fixed keywords. Hence, in the second experiment, the effects of GP become much significant.

Figures 9-13 show the retrieved result of the first generation in the second experiment. In first generation, the query is generated randomly. The generated query is "-freebsd+dream|doraemon-Gundam|mp3|cdrom". Figures 14-17 show the retrieved results after the fifteenth generation in the second experiment. The generated query is "mysql|network|mp3|car|freebsd". As experimental results show that in fifteenth generation, the keywords more relevant to the user's profile than the first generation, thus using GPWS to generate queries can get better performance than to generate queries randomly.

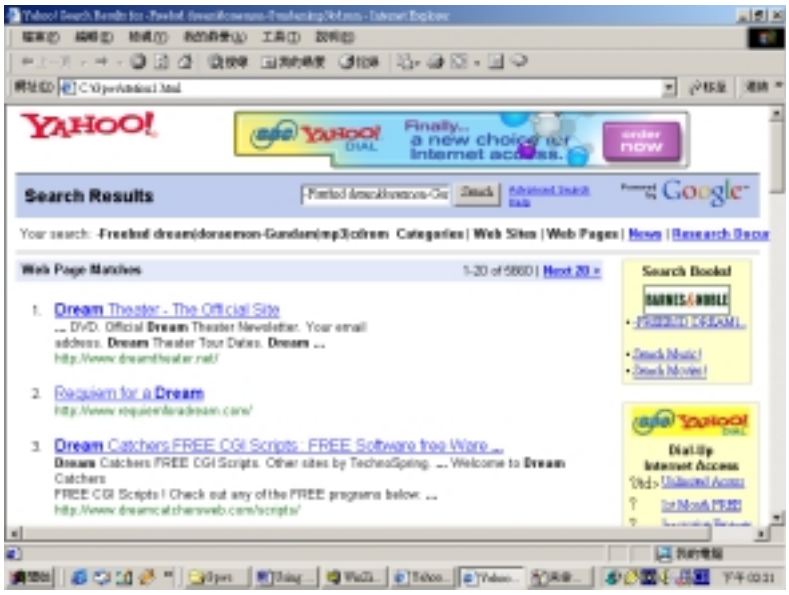

Figure 9. The retrieved Html of first generation-page1

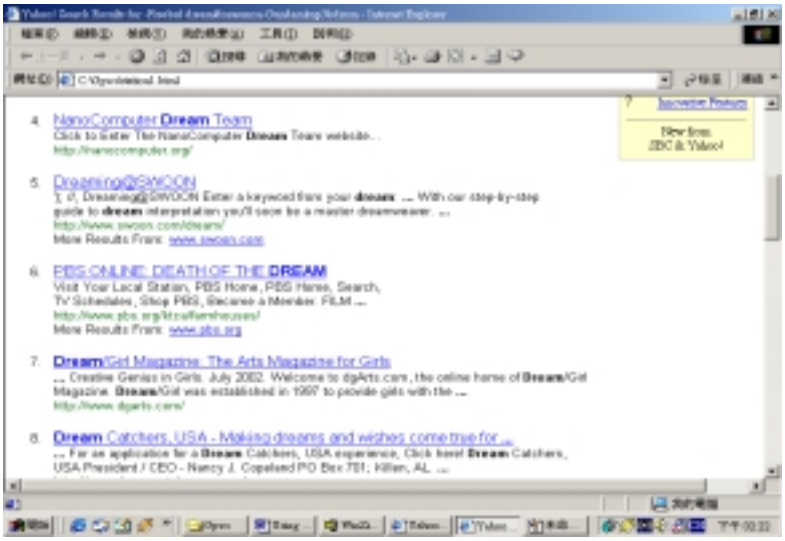

Figure 10. The retrieved Html of first generation-page2

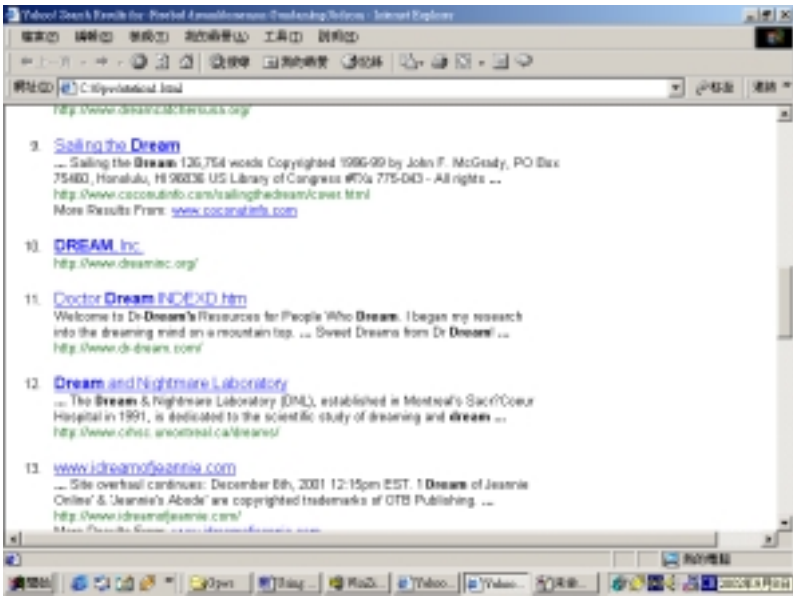

Figure 11. The retrieved Html of first generation-page3

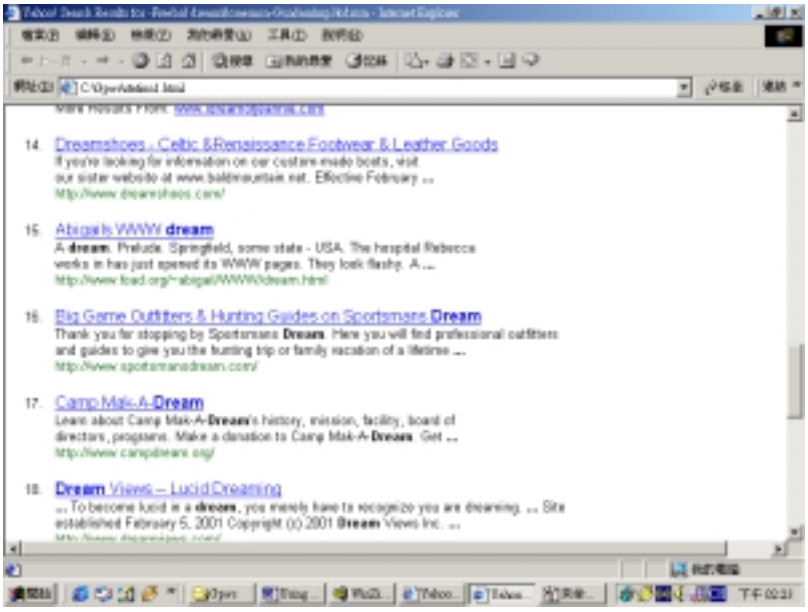

Figure 12. The retrieved Html of first generation-page4

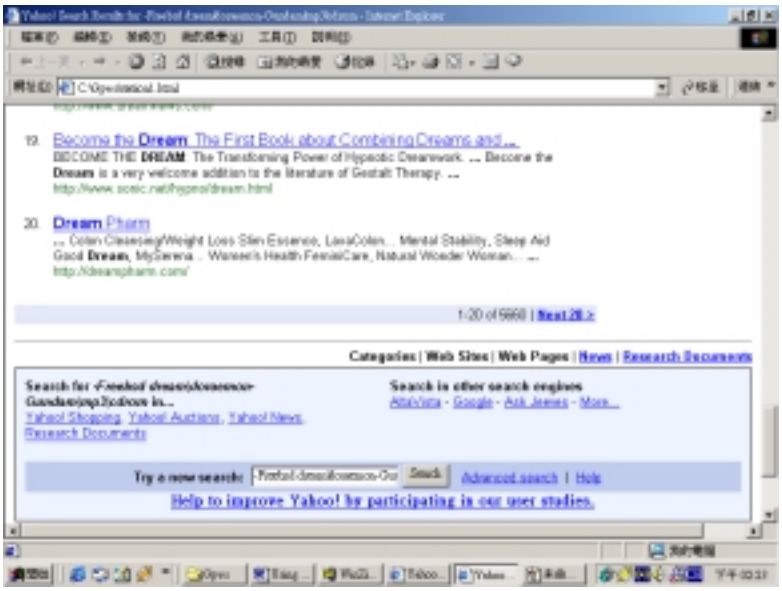

Figure 13. The retrieved Html of first generation-page5

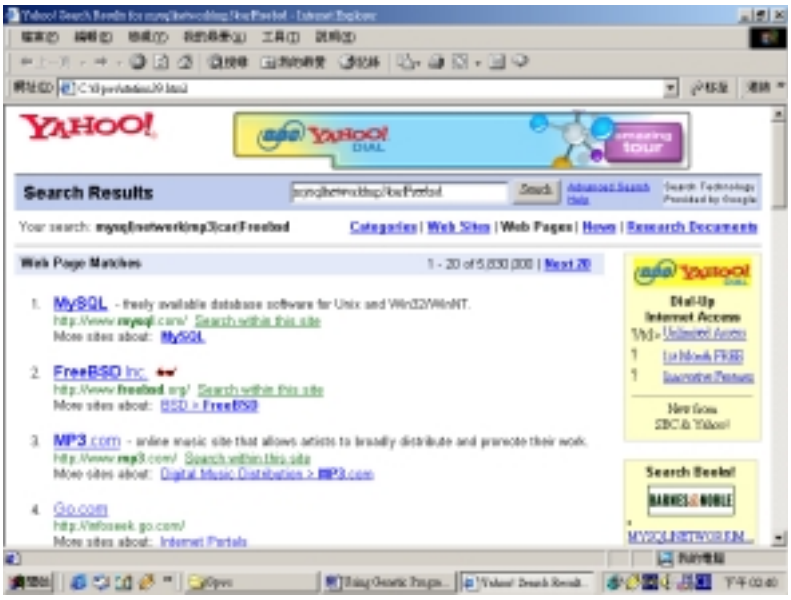

Figure 14. The retrieved Html of fifteenth generation-page1

Figure 15. The retrieved Html of fifteenth generation-page2

Figure 16. The retrieved Html of fifteenth generation-page3

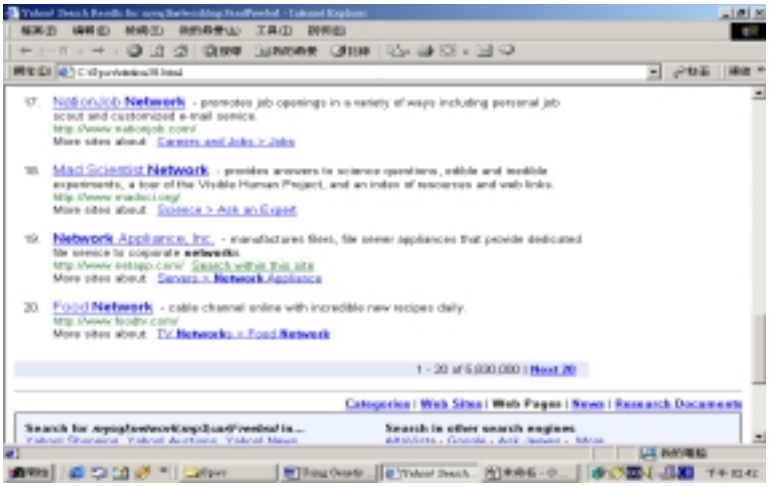

Figure 17. The retrieved Html of fifteenth generation-page4

5. Conclusions

We have proposed a genetic programming web search system (GPWS) to collect a user's behavior during Web surf, generate exact queries according to the user's interests, filter results and remove the redundancy. The results are displayed on a uniform user interface. As experimental results show that using GPWS to generate queries can get better performance than those of randomly generated queries. The degree of relevant of results and user's interests are increased. The results that retrieved using GPWS to generate the queries are more relevant to the user's interests.

References

[1] Z. Z. Nick and P. Themis, “Web Search Using a Genetic Algorithm,” *Journal of IEEE Internet Computing*, pp. 18-26, 2001.

[2] S. Aggarwal, F. Hung, and W. Meng, “WIRE--- A WWW-based Information Retrieval and Extraction System,” *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pp. 887–892,1998.

[3] C. Faloutsos and D. Oard, “A Survey Of Information Retrieval and Filtering Methods,” Iniversity Of Maryland college Park CS-TR-3514, 1995. Also available at Http://www.enee.umd.edt/medlab/filter/papers/survey.ps

[4] T. K. Shih, “Agent Communication Network-A Mobile Agent Computation Model fro Internet Applications,” *Proceedings of the IEEE International Symposium on Computers and Communications*, pp. 425-431, 1999.

[5] X. H. Zhang, H. Y. Wang, and H. Zhao, “An Autonomous System-based Distribution System for Web Search,” *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 1, pp. 435-440, 2001.

[6] K. L. Hung, “Design and Construction of Learning Purchasing Agent on the Internet,” *Proceedings of the National Computer Symposium*, Vol. 1, pp. 97-102, 2001.

[7] L. M. Stephens and M. N. Huhns, “Consensus Ontologies- Reconciling the Semantics of Web pages and Agents,” *IEEE Internet Computing*, Vol. 5, Issue 5, pp. 92-95, 2001.

[8] M. Brameier and W. Banzhaf, “A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining,” *IEEE Transactions on Evolutionary Computation*, Vol. 5, No.1, 2001.

[9] R. Jain and F. Anjum, “Mobile Agents for Personalized Information Retrieval: When Are They A Good Idea?” *Proceedings of Wireless Communications and Networking Conference*, Vol.1, pp. 242–245, 2000.

[10] S. Murugesan, “Intelligent Agents on the Internet and Web,” *Proceedings of the IEEE International Conference on Global Connectivity in Energy, Computer, Communication and Control*, Vol. 1, pp. 97-102, 1998.

[11] J. R. Koza, *Genetic Programming*, 3rd Printing, MIT press, 1993.

[12] X. Li, and W. B. Croft, “Evaluating Question-Answering Techniques in Chinese,” *Proceedings of HLT* 2001, pp. 201-206, 2001.

[13] D. Lawrie, W. B. Croft, A. Rosenberg, “Finding Topic Words for Hierarchical Summarization,” *Proceedings of SIGIR 01 conference*, pp. 349-357, 2001.

[14] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes- Compressing and Indexing Documents and Images*, 2nd Edition, Morgan Kaufmann Publishers, 1999.

[15] M. R. Henzinger, “Hyperlink Analysis for the Web,” *IEEE Internet Computing*, Vol. 5, Issue 1, pp. 45-50, 2001.

[16] B. G. Silverman, M. Bachann, and K. AI-Akharas “Do What I Mean: Online Shopping with a Natural Language Search Agent,” *IT Professional*, Vol. 16, Issue 4, pp. 48-53, 2001.

[17] J. Williams and R. Starzl, “Tuning up the Search Engine,” *IEEE Internet Computing*, Vol. 3, Issue 3, pp. 60-62, 2001.

[18] V. Thong, P. J. Moreno, B. Logan, B. Fidler, K. Maffey, and M. Moores, “Speechbot: an Experimental Speech-based Search Engine for Multimedia Content on the Web,” *IEEE Transactions on Multimedia*, Vol. 4, Issue 1, pp. 88-96, 2002.

[19] R. Jain, F. Anjum, and A. Umar, “A Comparison of Mobile Agent and

Client-Server Paradigms for Information Retrieval Task in Virtual Enterprises," *Proceedings of Academia/Industry Working Conference*, pp. 209 –213, 2000.

[20] S. Papastavrou, G. Samaras, and E. Pitoura, "Mobile Agents for World Wide Web Distributed Database Access," *IEEE Transactions on Knowledge and Data Engineering*, Vol.12, Issue: 5, pp. 802–820, 2000.

[21] P. Bellavista, A. Corradi, and C. Stefanelli, "Mobile Agent Middleware for Mobile Computing," *Computer*, Vol. 34, Issue: 3, pp. 73–81, 2001.

[22] C. W. Chiang, "A Skeleton Supporting Group Collaboration, Load Distribution, and Fault Tolerance for Internet-based Computing," Doctorial Thesis Of National Sun Yat-Sen University Taiwan, ROC, 2001.

[23] S. Wu and C. C. Liao "Virtual Proxy Servers for WWW and Intelligent Agents on the Internet," *Proceedings of the Thirtieth Hwaii International Conference on System Sciences*, Vol. 4, pp. 200-209, 1997.

[24] M. R. Henzinger, "On Near-Uniform URL sampling," *Proceedings of the Ninth International World Wide Web Conference*, pp. 295-308, 2000.