# Building Software Agents for Information Filtering on the Internet: A Genetic Programming Approach*

Byoung-Tak Zhang, Ju-Hyun Kwak, and Chang-Hoon Lee
Department of Computer Science and Engineering
Kon-Kuk University, Seoul 133-701, Korea
E-mail: {btzhang, jhkwak, chlee}@ai.konkuk.ac.kr
Phone: +82-2-450-3510 Fax: +82-2-446-9520

**Abstract**

With the explosive growth of information sources on the Internet, a number of software agents or softbots have been built to help users search and retrieve electronic documents from servers around the world. Information preferences vary greatly across users and therefore softbots must be highly personalized to serve the information filtering needs of the user. In this paper we formulate information filtering as a learning problem and develop a genetic programming approach to this problem. Given a set of sample documents, genetic programming evolves a population of agents that specialize to user interests. Our experimental results indicate that genetic programming is a promising technique for discovering and adapting to the specific interests of individual users.

**Keywords:** genetic programming, machine learning, softbots, Internet, information filtering.

*In *Proc. Int. Conf. on Genetic Programming (GP'96-Stanford)*, MIT Press, July 1996.

# 1 Introduction

The Internet is one of the largest publicly available "databases" of documents. With the Internet having seen an explosive growth in recent years, a number of services have arisen on the Internet to help users search and retrieve documents from servers around the world. Software robots or sotfbots are autonomous agents that interact with real-world software environments such as operating systems and the World Wide Web [1]. The softbot accepts goals in a high-level language, generates and executes plans to achieve these goals, and learns from its experience. The softbot enables a human user to access a wide range of information resources by stating what he or she wants accomplished. The softbot disambiguates the request and dynamically determines how and where to satisfy it.

Information preferences vary greatly across users and therefore softbots must be highly personalized to serve the specific individual interests of the user. The system should be able to select the articles deemed to be interesting to the user and eliminate the rest. Since filtering involves repeated interactions with the user, the system should be able to identify patterns in the users behavior. The filtering system must infer the habits of the user and specialize to them, i.e. recommend as many relevant articles and as few irrelevant articles as possible[9].

In this paper we describe an application of genetic programming to the construction of software agents. In contrast to linear-string genetic algorithms, genetic programming evolves individuals of variable-sized trees [4]. The individuals evolved by genetic programming in this work are agents that filter electronic documents on the Internet according to the specific interests of the user. In Section 2 we review related work. Section 3 develops the genetic programming approach to building information filtering agents. The last two sections describe and discuss the simulation results.

# 2 Software Agents for Information Gathering

Several software agents or softbots have been built to assist users in accessing information sources on the Internet. Examples are St. Bernard and Rodney [1], Maxims [6] and Newt [9], among others.

St. Bernard is a softbot that is able to locate and retrieve files based on a partial description of their characteristics. Rodney is a general-purpose UNIX softbot. Rodney enables a user to specify what to accomplish, in a high-level goal language, leaving the decision of how to accomplish it to the system. The softbot uses UNIX commands such as `mv`, `cd`, or `lpr` as operators to accomplish UNIX tasks such as "change the protection of file1 to readable".

Maxims is an agent that assists the user with email. Maxims learns to prioritize, delete, forward, sort and archive mail messages on behalf of the user. The agent continuously "looks over the shoulder" of the user as the user deals with email. As the user performs actions, the agent memorizes all of the situation-action pairs generated. When a new situation occurs that can be due to the user taking an action or due to some external event such as a message arriving, the agent compares the new situation with the memorized situations and tries to suggest an action which is based on the closest match.

Newt is a system that helps the user filter Usenet Netnews. A user can create one or many news agents and train them by means of examples of articles that should or should not be selected. An agent is initialized by giving it some positive and negative examples of articles to be retrieved. The user can also program the agent explicitly and fill out a set of templates of articles that should be selected. Once an agent has been bootstrapped, it starts recommending articles to the user. The user can give it positive or negative feedback for articles. Using the feedback the probability is adjusted that the agent will recommend similar articles in the future.

Softbots for information gathering on the Internet has to solve at least two problems in common. The one concerns where and how the relevant documents are retrieved. The other is to determine the preference of documents according to the current specific interests of the user. In the following, we will refer the software agents responsible for each of these tasks to a search agent and a filtering agent. In a full-fledged version, these two agents cooperate each other as follows.

1. Initialize the knowledge bases of the search agent and the filtering agent.
2. The search agent uses his knowledge base to search for documents on the Internet.
3. The filtering agent uses his own knowledge base to give preferences to the articles suggested by the search agent.
4. The user reads the articles in a decreasing order of preference and gives relevance feedback for each document.
5. The filtering agent learns from the relevance feedback and adapt his knowledge base.
6. Go to step 2.

The present work is focused on the document filtering task (steps 3 to 5). The search for documents (step 2) is simulated by the user at the moment. In the next section we describe a genetic programming method for evolving a population of filtering

agents. Though in a different context, a similar approach has been used by Kraft et al. [5]. They used genetic programming to build queries for retrieving abstracts from a database of journal issues. This is a nice example of using genetic programming for information retrieval.

In our setting, the set of sample documents changes as new documents are retrieved and the interests of the user changes over long periods of time. The adaptive nature of information filtering is one of its characteristics that distinguishes it from information retrieval [8, 9]. This setting is similar to the environment of the Newt system described above. Newt employs a genetic algorithm to explore the space of Netnews using a population profiles, each consisting of a fixed number of fields such as author and newsgroup. In our genetic programming approach the population consists of filtering agents or variable-sized trees having keywords at their terminal nodes. Genetic programming seems to allow more flexibility than the linear-string genetic algorithms in describing the content of documents and in adapting to changing interests of users. The next section provides a more detailed description.

# 3 Evolving Document Filtering Agents

The set of documents, $D$, of size $N$ represents the current interests of the user. Each document $D_j$ is labelled with relevance feedback $r_j$.

$$D = \{(D_1, r_1), (D_2, r_2), \cdots, (D_j, r_j), \cdots, (D_N, r_N)\} \quad (1)$$

The relevance feedback is either positive (+1) or negative (-1) and determined by the user.

$$r_j = \begin{cases} +1 & \text{if } D_j \text{ matches the current user interests.} \\ -1 & \text{if } D_j \text{ does not match the user interests.} \end{cases} \quad (2)$$

The positive feedback means that the document is a positive example of the current interests of the user. The negative feedback indicates the document is a counter-example of user's current interests.

A tree-shaped chromosome is used to represent the knowledge or rules of a document filtering agent. The terminal set of trees consists of keywords for describing the documents. The function set of trees consists of logical operations such as conjuctions, disjunctions, and negations. When the terminal nodes are instantiated by a specific document, the evaluation of the tree results in a two-way classification, either true (positive) or false (negative).

If the rule $A_i$ evaluates a positive document $D_j$ as positive, or a negative document as negative, then the agent $i$ is said to hit the user interests in the document $j$

and given a score of one, denoted by $h(i,j) = 1$. If the agent $i$ does not hit the user interests with respect to the document $j$, it is given no score, $h(i,j) = 0$.

$$h(i,j) = \begin{cases} 1 & \text{if } D_j \text{ is a positive example and } A_i \text{ evaluates it to positive} \\ & \textit{or} \text{ if } D_j \text{ is a counter-example and } A_i \text{ evaluates it to negative} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The goodness of agent $i$ is measured by the total number $H(i)$ of its hits with respect to the document set of size $N$:

$$H(i) = \sum_{j=1}^{N} h(i,j) \quad (4)$$

where the index $j$ runs over the documents. Agent $i$ is said to be better specialized to the user interests than agent $k$ if rule $A_i$ hits more documents in $D$ than rule $A_k$, i.e.

$$A_i \text{ is better than } A_k \text{ if } H(i) > H(k) \quad (5)$$

The fitness of an agent is defined as:

$$f(i) = \frac{\text{Num. of hits by agent } i}{\text{Total number of documents}} = \frac{H(i)}{N} \quad (6)$$

This measure is equivalent to the hit ratio, the fraction of the correctly filtered documents to the number of all documents in the current document set.

# 4 Experimental Results

In simulations, 100 documents were used for evolving agents. The documents were HTML (HyperText Markup Language) articles or their excerpts retrieved from the World Wide Web using Netscape. The test domain was the evolutionary computing area. Each document was given a relevance feedback. A half of the documents were positive examples and another half were negative examples.

Keywords used as terminals are listed in Table 4. The function set consisted of three logical operations: `AND`, `OR`, and `NOT`. Crossover exchanged the subtrees of two parents, selected at random. Mutation changed the terminals and functions at a rate of 0.03 for each node. Elitist strategy was used in combination with proportionate selection so that the best individual always survives the selection. Maximum depth of the tree was 10 at initialization.

Some agents evolved by genetic programming are depicted in Figure 1. The rectangles represent the terminal nodes and the ovals denote the logical functions. These agents achieved a 97% performance for the sample data. Figure 2 shows the

| | | | |
|---|---|---|---|
| genetic | algorithm | strategy | programming |
| evolution | selection | generation | fitness |
| individual | chromosome | crossover | mutation |
| function | EA | GA | GP |

Table 1: Sample keywords used for filtering documents on evolutionary computing.

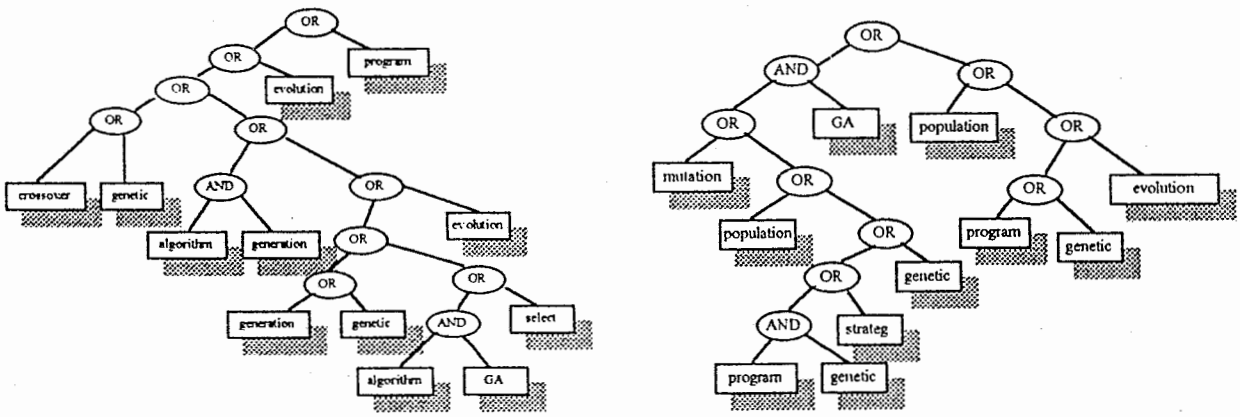


Figure 1: Examples of evolved agents.

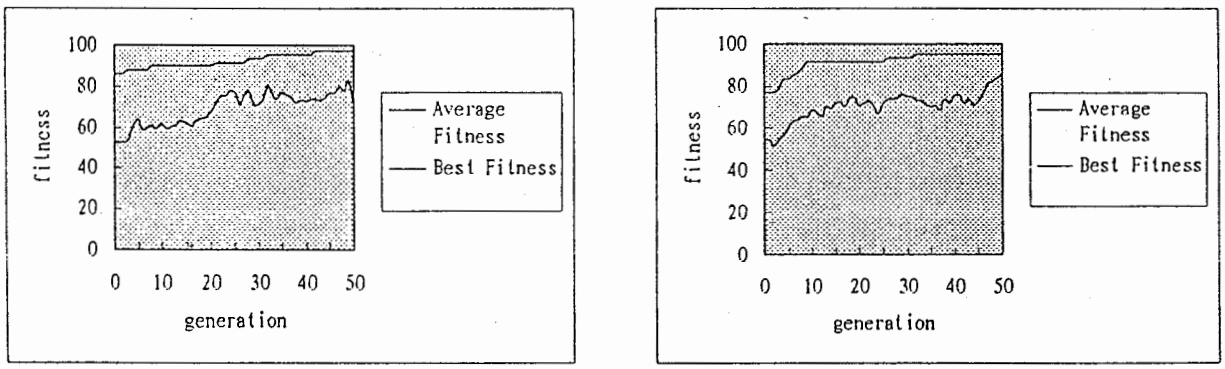


Figure 2: The best and average fitness for each generation: two example runs.

| pop size | max gen | succ critr | succ runs |
|---|---|---|---|
| 50 | 50 | 0.90 | 80% |
| | | 0.95 | 30% |
| 100 | 50 | 0.90 | 90% |
| | | 0.95 | 30% |
| 150 | 50 | 0.90 | 100% |
| | | 0.95 | 10% |
| 200 | 50 | 0.90 | 100% |
| | | 0.95 | 40% |

Table 2: Number of successful runs for different combinations of population size and success criterion. The number used for success criterion is the fitness value after the max-gen number of generations.

best and average fitness of the populations, scaled to the interval of 0 and 100. Using population size of 100, most of the runs evolved agents with higher than 90% performance within 50 generations. We also investigated the behavior of the algorithm with increasing size of population. As Table 2 shows, the fraction of runs with reasonalbe performance (90% or higher) to the total number of trial runs, increases as the population size grows.

# 5 Summary and Future Work

We have described a genetic programming method for evolving software agents that filters HTML documents based on the user interests. The specific interests are represented by a set of sample documents which reflect the preferences of the user. The goal of genetic programming was formulated to evolve the rules for filtering postive/negative interests using keywords as terminals and logical operators as functions. The preliminary experiments indicate that the genetic programming methode is useful in evolving the filtering rules specialized to user interests.

One difficulty found in the experiments was the explosive growth of the individual structures and thus slowed down evolution speed when a large number of generations are allowed in combination with a large terminal set. This is a typical example of the necessity of Occam's razor for improving evolution speed. We are now incorporating into this task the adaptive fitness function developed in [11] and [12].

With respect to the application domain, future work includes two extensions. One is to devise a method that dynamically extends the keywords for the description of documents. This will enable the representation to change fluently as the user interests change. Another extension involves continuous-valued, instead of binary,

rating of selected documents. This will give the user more convenience in determining the sequence of reading the suggested documents. To this end, a mechanism seems necessary that produces a real-valued preference when the tree is evaluated.

# References


[1] O. Etzioni, N. Lesh, and R. Segal, Building softbots for UNIX, Technical Report, Dept of CSE, University of Washington, 1992.

[2] O. Etzioni and D. Weld, A softbot-based interface to the Internet, *Communications of the ACM*, 37(7):72-76, 1994.

[3] J. Horng, B. Liu, and C. Kao, A genetic algorithm for database query optimization, In *Proc. of IEEE Conf. on Evolutionary Computation*, IEEE Press, 1994, pp. 350-355.

[4] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[5] D. Kraft, F. Petry, B. Buckles, and T. Sadasivan, The use of genetic programming to build queries for information retrieval, In *Proc. of IEEE Conf. on Evolutionary Computation*, IEEE Press, 1994, pp. 468-473.

[6] Y. Lashkari, M. Metral, and P. Maes, Collaborative interface agents, In *Proc. AAAI-94*, MIT Press, Cambridge, MA, 1994.

[7] P. Maes, Agents that reduce work and information overload, *Communications of the ACM* 37(7):31-40, 1994.

[8] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Reading, MA: Addison-Wesley, 1989.

[9] B. D. Sheth, A learning approach to personalized information filtering, Masters Thesis, Dept. of EECS, Massachusetts Institute of Technology, February 1994.

[10] J.-J. Yang and R.R. Korfhage, Query modification using genetic algorithms in vector space models, Technical Report LIS045/IS92001, School of Library and Information Science, University of Pittsburgh, Pittsburgh, PA, 1992

[11] B. T. Zhang and H. Mühlenbein, Balancing accuracy and parsimony in genetic programming, *Evolutionary Computation*, 3(1): 17-38, 1994.

[12] B. T. Zhang and H. Mühlenbein, Adaptive fitness functions for dynamic growing/pruning of program trees, *Advances in Genetic Programming 2*, Chapter 12, Cambridge, MA: MIT Press, 1996.