International Journal of Computational Intelligence and Applications
Vol. 5, No. 4 (2005) 471–493

Imperial College Press
www.icpress.co.uk

SAGAXSEARCH: AN XML INFORMATION RETRIEVAL MECHANISM USING SELF ADAPTIVE GENETIC ALGORITHMS

K. G. SRINIVASA*, S. SHARATH† and K. R. VENUGOPAL‡

Department of Computer Science and Engineering
University Visvesvaraya College of Engineering
Bangalore University, Bangalore — 560001, India
**kgsrinivas@msrit.edu*
†*sharaths@infosys.com*
‡*venugopalkr@gmail.com*

M. PATNAIK

Microprocessor Applications Laboratory
Indian Institute of Science
Bangalore — 560012, India
lalit@micro.iisc.ernet.in

Received 19 June 2005
Revised 14 October 2005

The XML technology, with its self-describing and extensible tags, is significantly contributing to the next generation semantic web. The present search techniques used for HTML and text documents are not efficient when retrieving relevant XML documents. In this paper, Self Adaptive Genetic Algorithms are presented to learn about the tags, which are useful in indexing. The indices and relationship strength metric are used to extract fast and accurate semantically related elements in the XML documents. The Experiments are conducted on the DataBase systems and Logic Programming (DBLP) XML corpus and are evaluated for precision and recall. The proposed SAGAXsearch outperforms XSEarch[3] and XRank[20] with respect to accuracy and query execution time.

Keywords: Self-Adaptive Genetic Algorithms; indexing; XML search; proximity metric.

1. Introduction

Extensible Markup Language (XML) has been recognized as a standard for describing the data format and its meaning. The user defined tags associate the semantics with the contents of XML documents. Hence XML is a medium for interoperability over the Internet. With these advantages, the amount of data that is being published on the Web in the form of XML is growing enormously and many naïve users find the need to search over large XML document collections. The keyword query is a search technique that does not require the users to know the structure of the underlying data. There is no need to learn complex query languages to discover

knowledge. Thus, the keyword search over XML documents in the present context is of significant importance.

Keyword search over large document collections has been extensively used for text and HTML documents[1] and it has two main drawbacks. First, search engines are not as intelligent as their users. For example, a keyword search "*Kevin Database Technology*" will retrieve documents in which *Kevin* is the author and also documents in which *Kevin* is mentioned in the references with equal priority, though the former is more semantically relevant to the user. The second drawback is that keyword queries are inherently flexible in nature and can produce large number of results. The results are of varying relevance to the user and they need to be ranked. The time taken to rank the results should be a small portion of the total query execution time. In contrast, a structured query language will retrieve only the most relevant results, but the complex query syntax makes it unsuitable for naïve users. Thus an approach which has the flexibility of keyword queries that still retains the accuracy of a query language would be most suitable. The keyword search over XML documents pose many new challenges.[2] First, the result of a search over XML documents is not the document in its entirety, but only relevant document fragments. As an illustration, consider a keyword search query "*operating system*" over the XML document in Fig. 1. The keyword is associated with the <*inproceedings*> tag (line 19) and is a relevant result, but returning only the elements related to the <*inproceedings*> tag (line 19) would be more intuitive than returning the whole document. Thus, granularity of the search terms must be refined when searching over XML document corpus.[3]

Second, the result of a keyword search over XML documents must be semantically interconnected document fragments. Consider the keyword search "*Ananth Synchronization mechanism*" over the XML document shown in Fig. 1. Though, the keywords exist independently in the XML document (line 4, 15), they belong to different <*inproceedings*> tags. The author "*Ananth*" is not semantically interconnected to the title "*synchronization mechanism*". Thus, only semantically interconnected document fragments should contribute to the search results.

Finally, XML documents include large amounts of textual information and part of this is rarely searched. Building a single index for the whole document will make the index bulky and difficult to manage. Thus, there is a prospect of improving the search accuracy and query execution time by separating the frequently searched tags from the occasionally searched ones and building separate indices for both, and this is explored by the use of a Self-Adaptive Migration Model Genetic Algorithm (SAMGA).[5] A search over the XML documents in the decreasing order of the importance of the tags is accurate and efficient.

Genetic algorithms (GA)[4,6] is an extremely effective technique for searching enormous, possibly unstructured solution spaces. The solutions are represented by *chromosomes* which are strings of *alleles.* The recombination of chromosomes creates new strings with alleles taken from the parent chromosomes. Since solutions are evolved by trying out answers and combining the answers that work best, the

```
 1) <dblp>
 2) <inproceedings>
 3) <author>I.S. Vipin</author>
 4) <author>C.G. Ananth</author>
 5) <author>G. Sarah</author>
 6) <title>Land Use: Problems and Experiences.</title>
 7) <pages>135-172</pages>
 8) <year>1979</year>
 9) <crossref>conf/ibm/1979</crossref>
10) <booktitle>Data Base Techniques </booktitle>
11) <url>db/conf/ibm/db79.htm#zaGM79</url>
12) < /inproceedings>
13) <inproceedings>
14) <author> A. N. Ravi</author>
15) <title>Synchronization mechanisms</title>
16) <pages>2-22</pages>
17) <year>1980</year>
18) <crossref>conf/ibm/1980</crossref>
19) <booktitle>Operating Systems </booktitle>
20) <url>db/conf/ibm/80.html#Saito80</url>
21) </inproceedings>
22) </dblp>
```

Fig. 1. Example of a XML document.

technique is particularly well-suited to solving problems where the solution space is large. In island or migration model of Genetic Algorithms instead of a single population, a set of populations are considered.[7] Regular migration of individuals between the populations ensures that the best offsprings are retained and less fit individuals are eliminated.

The human search strategy which is efficient for small documents, is not viable when performing search over enormous amounts of data. Hence, making search engines cognizant of the search strategy using GA, can perform fast and accurate search over large document collections.

Contributions: We have explored the possibility of retrieval and ranking of XML fragments based on keyword queries. Self adaptive and real coded Genetic Algorithms are used for learning tag information. A measure of distance metric between the keywords among the XML documents is proposed. Genetically learned tag information is used to retrieve semantically interconnected document fragments.

The rest of the paper is organized as follows. Section 2, describes the motivation behind the proposed technique with an example. The related work in the field of information retrieval from XML documents is presented in Sec. 3. The XML data

model is discussed in Sec. 4. The genetic learning of tag information is explained in Sec. 5. In Sec. 6, we discuss the XML identification schemes and the search technique. Section 7 presents the experimental results and finally, Sec. 8 concludes the paper with directions for future work.

2. Motivation

Consider the XML document fragments, an excerpt from a health-care record. Consider a keyword search "*Vinu salbutamol*" over the XML document in Fig. 2. A standard HTML search engine would consider the whole document in Fig. 2 as a suitable response, due to the presence of both the terms in the search query. However, in XML environment the two search terms occur as totally unrelated elements in the document as they belong to the medical records of different patients.

In the XML document of Fig. 2 the keyword "*penicillin*" appears in two different contexts; first it is associated with the <*administer*> tag and then with the <*drug_allergy*> tag and the tag name precisely categorizes between the two occurrences. Additional information like name, record identifiers are also explicitly captured using application specific self explanatory tags. This is useful in keyword search over XML documents. Thus exploiting the tagged and nested structure of XML can help in effective knowledge discovery. We describe in this paper, an architecture, implementation and evaluation of a search engine for retrieving relevant XML document fragments in real time.

3. Related Work

Search engines for XML can be classified into two general categories: database-oriented and information retrieval-oriented. In the database approach,[11] the XML

```
<medical_records>
<patient>
<name> Vinu Krishnan </name>
<record_id> 4312</record_id>
<administer> penicillin </title>
<drug_allergy>none</allergies>
</patient>
<patient>
<name> Victor James </name>
<record_id> 4313</record_id>
<administer>salbutamol </administer>
<drug_allergy>penicillin</drug_allergy>
</patient>
</medical_records>
```

Fig. 2. Example health-care record.

documents are decomposed and stored in relational database. However, query processing becomes expensive since, in many cases, an excessive number of joins is required to recover information from the fragmented data. Object-oriented databases have been associated with XML document collections.[9] In this case, retrieval of information from XML documents is considered as an object view problem.

Extensive research has been done on structured declarative queries over XML documents. A structured declarative query is supported by XQuery,[12] which is analogous to SQL queries over relational databases. Though XQuery can achieve perfect precision and recall, they require user to learn query semantics and in cases where the user is unaware of the document structure, a search cannot be performed. An improvement over XQuery that has elegant syntax and semantics is developed in Ref. 13.

Information retrieval techniques can consider XML documents as normal text documents, with additional markup overhead. There are several ways of handling the tags. For simplicity, the tags can simply be ignored but the document loses its semantics, leading to lower retrieval performance. When tags are taken into consideration, search can retrieve documents containing certain tags, or certain words. Keyword search over XML documents falls under this category.

In information retrieval, Genetic Algorithms have been used in several ways[14] but in a different context. Genetic Algorithms have been used to modify user queries[15,16] and for automatic retrieval of keywords from documents. In Ref. 17, GA is applied to adapt multiple matching functions obtained from the combination of scores using individual matching functions. This is used to rank and retrieve documents. In Ref. 18, GA has been used for mining of HTML structures. The algorithm learns the important factors of HTML tags through a series of queries.

Keyword search over XML documents is supported by XKeyword,[19] XRANK,[20] and XSEarch.[3] All these keyword search techniques have elaborate ranking schemes. The simplicity of the search queries, i.e. keywords, make these techniques suitable for naïve users. But, precision and recall values tend to suffer and the extensive ranking function employed acts as an overhead during query execution.

In XRANK,[20] the hierarchical and hyperlinked structure of XML documents are taken into account while computing the ranks for the search results. A ranking technique at the granularity of XML elements is considered here. XRANK can query over a mix of XML and HTML documents.

XSEarch[3] introduces a concept known as *interconnected* relationship. However, checking for the interconnected relationship is a huge overhead during runtime. Moreover, XSEarch suffers from drawbacks similar to other keyword search engines: unimpressive precision and recall values. In our proposed SAGAXsearch algorithm, the association of Self Adaptive Genetic Algorithms with keyword queries ensures high accuracy, i.e. very few non-relevant fragments (high precision) and most of the relevant fragments (high recall) will be selected as results. This is indicated by the experimentation results in Sec. 7.

4. XML Data Model and Query Semantics

In this section, we briefly describe the XML data model and the keyword query semantics for search over XML documents.

4.1. *Data model*

The Extensible Markup Language (XML) is a human readable, machine understandable, general syntax for describing hierarchical data, applicable to a wide range of applications. XML allows users to bring multiple files together to form a compound document. The XML document consists of nested elements starting from the root and corresponding associated values. Figure 1 shows a sample XML document, an excerpt from the XML version of DBLP. The <*dblp*> is the root element, and it has <*author*>, <*title*>, <*pages*>, <*year*> as its sub-elements. The XML document can be considered as a directed, node-labeled *data graph* $G = (X, E)$. Each node in X corresponds to an XML element in the document and is characterized by a unique *object identifier*, a *label* that captures the semantics of the element and leaf nodes are associated with a sequence of *keywords*. E is the set of edges which define the relationships between nodes in X. The edge $(l, k) \in E$, if there exists a directed edge from node l to node k in G. The edge $(l, k) \in E$ also denotes that node l is the *parent* of node k in G. Node l is also the ancestor of node k if a sequence of directed edges from node l leads to node k. The document tree for the XML document in Fig. 1 is shown in Fig. 3.

4.2. *Query semantics and results*

Let the XML document tree in Fig. 3 be called τ. Let x be an interior node in this tree. We say that x directly satisfies a search term k if x has a leaf child that contains the keyword k and x indirectly satisfies a keyword k if some descendent of x directly satisfies the search term k. A search query $q = \{k_1, k_2, \ldots, k_m\}$ is satisfied

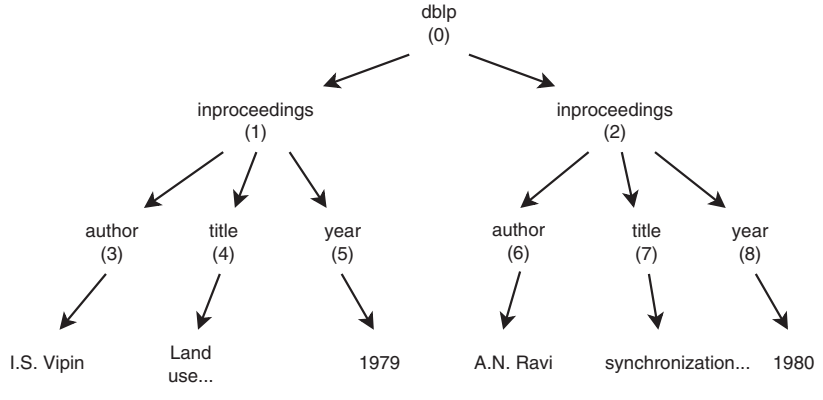

Fig. 3. Part of the XML document tree.

by a node x iff x satisfies each of $k_1, k_2, \ldots, k_m$ either directly or indirectly. For example, in the XML tree shown in Fig. 3, *inproceedings* (1) satisfies the search term "*Vipin*" and the search term "*Vipin 1979*" but not the term "*Vipin 1980*".

We say that $R = \{x_1, x_2, \ldots, x_l\}$ is a search result consisting of l nodes if:

(i) Each node in the set $\{x_1, x_2, \ldots, x_l\}$ satisfies at least one term from $\{k_1, k_2, \ldots, k_m\}$.
(ii) $\{x_1, x_2, \ldots, x_l\} \in \tau$.
(iii) The set of nodes $\{x_1, x_2, \ldots, x_l\}$ shall be semantically relevant.

Semantically related nodes are nodes that appear in the same context; for example, an author and the title of his book having the *inproceedings* ancestor node. A mathematical measure of semantic relationship is given in Sec. 6. The various steps in the working of SAGAXsearch are enlisted below:

1. A representative training set is chosen to assist the genetic learning of tags.
2. The keyword queries and the relevant search results are collected from the user.
3. The genetic algorithm retrieves the tag combination, which can answer a maximum number of training queries.
4. Separate indices are built for the frequently used and occasionally used tag combinations.
5. A search over the XML documents in the decreasing order of importance of tags is performed.
6. The search produces only semantically related results.

5. Genetic Learning of Tags

XML documents include extensible tags for formatting the data. The tags represent the semantics of the data and thus can contribute to improve the accuracy of keyword search. Making use of Genetic Algorithms to learn the tag information has two advantages. First, separate indices can be built for both the frequently and less frequently searched tags. The problem space in such an operation is large, as the XML document corpuses are usually huge. Hence, Genetic Algorithms are used because of their ability to find accurate solutions in large problem spaces. Second, when the same keyword appears more than once in the XML document with different semantics (different tags), the knowledge learnt from the GA is used to rank the search results. Hence, the results that are more relevant to the user queries are better ranked than the other results. The architecture of the genetic learning system is illustrated in Fig. 4.

GA is an evolutionary process where at each generation, from a set of feasible solutions, individuals are selected such that those with higher fitness value have a greater possibility of reproduction. At each generation, the chosen individuals undergo crossover and mutation to produce populations of successive generations. The *selection* chooses the best individuals for crossover. With *crossover*, the characteristics of the parents are inherited by the individuals in the next generation.

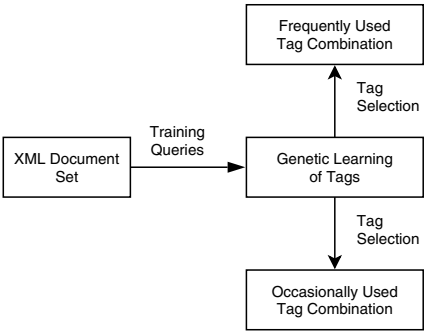

Fig. 4. Genetic learning of tags.

Mutation helps in restoring lost or unexplored regions in the search space. These three operators are inspired from the biological process of evolution and can find possible solutions even in a large problem space.

In simple GA, the three basic operators of GA namely, selection, crossover and mutation are fixed *a priori*. As the individuals evolve through generations, these operators remain constant. A new breed of GA called adaptive GA[4] adjusts the values of the operators based on the fitness of the individual in the population. Such an adaptive GA can exploit previously discovered knowledge for a focused search on parts of the search space which is more likely to yield better results and at the same time can search over the unexplored regions. In migration model GA,[7] instead of a single population, a set of populations is evolved. The basic operators of GA are applied independently for each population and at some regular intervals; individuals are exchanged between populations for a more diversified search.

In order for the self adaptive real coded GA to learn the tag information, it has to be adaptive in three aspects. The first parameter that is adaptively changed is the size of each population. The population size is determined by the fitness of the best individual in the population compared to the mean fitness of the population. The number of individuals in the population P_i is updated as,

$$n_{i,t+1} = n_{i,t} + (f(P_i)/\bar{f}) - 1,$$

where t is used to represent the time in generations. With this update, the size of the population grows when the fitness is greater than the value of the mean fitness and vice versa. Thus, the algorithm is more explorative in the problem space where there is more likelihood of finding the solution. Though the number of individuals in each population varies, the total number of individuals in the ecosystem remains the same.

The second parameter that is dynamically updated is the mutation rate and is given by,

$$pm_{i,t+1} = pm_{i,t} + (\bar{n}/n_i - 1) \times 0.0001.$$

Using this update, we see that, if the number of individuals in a population is less than the size of the mean population then the mutation rate is increased in order to make the search more explorative. In contrast, if the size of the mean population is smaller, then the mutation rate is decreased.

The final parameter that is adaptive in the algorithm is the rate of migration. Migration refers to copying individuals from one population to another. Migration helps in discovering new schemas generated by the crossover of two populations. In the algorithm, migration occurs only when the average fitness of the populations remains unchanged between two generations. Thus, when populations have attained a steady state, migration occurs to try and discover a new schema.

The selection operator tries to improve the quality of the future generations by giving individuals with higher fitness, a greater probability of getting copied into the next generation. Here the assumption is that parents with higher fitness values generate better offspring. The purpose of SAMGA is to select from the tag pool, the tag combinations which are interesting to a user. The user has to first issue a set of search queries $q = \{k_1, k_2, \ldots, k_m\}$. The documents satisfying the search terms are retrieved as results. The user has to classify the results relevant to him. This is the feedback given to the system in order to learn the user interest. The fitness function used in the GA is given by,

$$\text{fitness} = \alpha \times \left(\sum_{i=1}^{N} \text{freq}(i)/\text{rank}(i) \right) + (1 - \alpha) \times N,$$

where N is the number of documents retrieved with a specific tag configuration, $\text{freq}(i)$ is the frequency of occurrence of the terms of the query $q = \{k_1, k_2, \ldots, k_m\}$ in the ith retrieved document. The retrieved documents are ranked according to the frequency of occurrence of the terms of the query. The $\text{rank}(i)$ denotes the rank of the ith retrieved document, provided the document is also classified as relevant by the user; α is a parameter that is used to express the degree of user preference for accuracy of the search results and the total number of documents that are retrieved. This fitness value is assigned to each individual of the population. The reproduction probability of the individuals in the population depends upon its fitness value and the fitness values of the other individuals in the population. The selection operator used in the algorithm is *stochastic universal sampling*. Here individuals of the population are assigned contiguous segments on a straight line based on their fitness values. Let b be the total number of individuals selected, which are placed on equidistant points over a line. The distance between the points is given by $1/b$. Such a selection scheme has a zero bias and minimum spread, and is found suitable for our algorithm.

The recombination operator used is intermediate recombination, where the variable values of the offspring are around and between the variable values of the parents. Geometrically intermediate recombination produces variables with a slightly larger hypercube than that defined by the parents but constrained by the values of ρ. A real valued mutation operation is also applied in the algorithm to explore new regions and make sure that good genetic material is never lost.

Consider a representative training set with n documents on which keyword search is to be performed. Let $q = \{q_1, q_2, \ldots, q_m\}$ be a collection of typical user queries where q_i represents the ith query and m is the total number of queries.

- $E = \{P_1, P_2, \ldots, P_{np}\}$ be the ecosystem with np sub-populations,
- $P_i[j]$ represents the jth individual of the population P_i,
- n_i be the size of the population P_i,
- $\bar{f}_i$ be the average fitness of population P_i,
- $f(P_i)$ be the fitness of the best individual of the population P_i,
- $\bar{f}$ be the average fitness of the ecosystem,
- $\bar{n}$ be the average number of individuals per population, and
- pm_i be the rate of mutation used for population P_i.

Self Adaptive Genetic Algorithms for Learning Tag Information:

1) **for** each population P_i in the ecosystem
   i. Set n_i, number of individuals in the population P_i to some arbitrary value n_0.
   ii. Assign random tag weights for every individual of P_i
   iii. Set the mutation rate pm_i for population P_i, to some arbitrary value pm_0.
2) **next**
3) **for** gen = 1 : maximum generation limit, **do**
   a) $nsum = 0$
   b) $fsum = 0$
   c) **for** each population P_i in the ecosystem
      i. Order the tags by their decreasing weights and select the top k tags for each individual in P_i.
      ii. Evaluate the fitness of all individuals in the population P_i using the top k tags and the training queries q. Evaluate $f(P_i)$ the best fitness of the population.
      iii. $nsum = nsum + n_i$
      iv. $fsum = fsum + f(p_i)$
   d) $prev = \bar{f}$
   e) $\bar{f} = fsum/np$
   f) $\bar{n} = nsum/np$
   g) **for** each population P_i in the ecosystem
      i. $pm_i = pm_i + (\bar{n}/n_i - 1) * 0.0001$
      ii. $n_i = n_i + (f(p_i)/\bar{f}) - 1$
      iii. **if** $(n_i == 0)$ delete population P_i

(*Continued*)

(*Algorithm Continued*)

h) **for** each population P_i in the ecosystem

i. For the population P_i perform a selection with *stochastic universal sampling* as the selection operator and with the modified population size n_i.

ii. Perform *discrete recombination* on the selected individuals of population P_i

iii. Perform mutation on the individuals of the population P_i with the mutation probability pm_i.

i) **next**

j) **if** $prev == \bar{f}$, migrate the best individuals between populations.

l) **endif**

4) **next**

5) **end**

The definition of the chromosome is represented as $j = \{j_1, j_2, \ldots, j_l\}$ where j_i, a real number, denotes the weight of the tag i, l is the total number of distinct tags appearing in the document corpus which can be determined from the Document Type Definition (DTD) of the corpus. Thus, a tag weight is associated with each of the distinct tags appearing in the document collection.

The result of this self adaptive GA is the classification of tags as either frequently used or occasionally used. This precise categorization helps in maintaining separate indices for the information within the tags. The information within the frequently used tags is stored in an index called Most Frequently used Index (MFI) and the information within the occasionally used tags is stored in an index called Less Frequently used Index (LFI). The problem can be generalized to create many indices, prioritized according to the frequency of their usage.

6. Search Algorithm

The response to a search over an XML document is not the document in its entirety but only semantically related and relevant document fragments. In this section we discuss the identification schemes and semantic relationship between nodes in the XML tree. An algorithm to retrieve and rank the results is also discussed.

6.1. *Identification scheme*

The granularity of search over XML documents is not at the document level, but at the node level in the XML document tree. Hence, an identification scheme for the nodes in the document tree is required. This is accomplished by encoding the position of each node in the tree as a data value before storing it in an index. Given the identification values of the nodes, the scheme must also be able to reconstruct

the original XML tree. An identification scheme called Hierarchical Vector for Identification (hvi) is derived.

Let x be a node in the XML document tree τ. Then the Hierarchal Vector for Identification of x is recursively defined as,

$$\text{hvi}(x) = [\text{hvi}(\text{parent}(x))j] \quad \text{and} \quad \text{hvi}(\text{root}) = \tau_{\text{id}}.$$

Here, τ_{id} is a unique identification number assigned to the XML document tree τ, parent(x) is the parent of the node x in the XML document tree τ and j represents the sibling number of the node x with respect to its parent. With this identification scheme, each node captures its absolute position within the whole document. The hvi of a node identifies itself and all its ancestors. The hvi of various nodes in two XML documents are shown in Figs. 5(a) and (b).

Theorem 1. *Let $\tau_1, \tau_2, \ldots, \tau_n$ represent the XML document trees of the documents with identification numbers $(1, 2, 3, \ldots, n)$, where n is the number of documents. Then, $\{\exists\ x_i \in \{\tau_1, \tau_2, \ldots, \tau_n\}\ \Lambda\ \exists\ x_j \in \{\tau_1, \tau_2, \ldots, \tau_n\} x_i \neq x_j, \text{hvi}(x_i) \neq \text{hvi}(x_j)\}$, i.e. there exist no two distinct nodes among all the XML documents in the collection, such that they have the same* hvi.

Proof.

Case 1. Consider the two nodes x_i and x_j are present in different documents. i.e. $x_i \in \tau_i$ and $x_j \in \tau_j$ such that $\tau_i \neq \tau_j$. Since $\tau_i \neq \tau_j$, $\tau_{\text{id}}(x_i) \neq \tau_{\text{id}}(x_j)$, $\text{hvi}(x_i) \neq \text{hvi}(x_j)$.

Case 2. Consider the two nodes x_i and x_j that are present in the same document. i.e.$\{x_i, x_j\} \in \tau$. Since both the nodes are in the same XML document $\tau_{\text{id}}(x_i) = \tau_{\text{id}}(x_j)$. But, since $x_i \neq x_j$ (from the statement of the theorem) and $\{x_i, x_j\} \in \tau$ there exist two possibilities $p(x_i) = p(x_j)$ or $p(x_i) \neq p(x_j)$. If $p(x_i) \neq p(x_j)$ then $\text{hvi}(x_i) \neq \text{hvi}(x_j)$. If $p(x_i) = p(x_j)$ then x_i and x_j represent different siblings of the same parent; therefore $\text{hvi}(x_i) \neq \text{hvi}(x_j)$. Thus, each element of all the XML documents in the document collection is assigned a unique identifier. From Figs. 5(a) and (b), it can be observed that there are no two nodes with the same hvi values. The same is true for a collection of n documents. □

6.2. *Relationship strength*

Let $\text{hvi}(x_i)$ and $\text{hvi}(x_j)$ represent the hvi of two distinct nodes x_i and x_j, existing in the XML document tree τ. The length of the longest common prefix (lcp) for both the hvi is denoted as $\text{lcp}(x_i, x_j)$. Consider two keywords k_1, k_2. The relationship strength between these two keywords, denoted as $\text{RS}(k_1, k_2)$ is defined as,

$$\text{RS}(k_1, k_2) = \text{lcp}(x_i, x_j) \,|\, (x_i \text{ directly satisfies } k_1, x_j \text{ directly satisfies } k_2).$$

The condition that the node should directly satisfy the keyword ensures that only those nodes satisfying the keyword and also having the longest length of their identification vectors (hvi), are selected while evaluating the Relationship Strength (RS).

This is important because a node and all its ancestors satisfy a keyword, but a true measure of RS is represented only by the node which directly satisfies the keyword.

Consider two keywords k_1 and k_2 such that x_i directly satisfies k_1, x_j directly satisfies k_2. If $x_i \in \tau_i$ and $x_j \in \tau_j$ such that $\tau_i \neq \tau_j$, then $\mathrm{RS}(k_1, k_2) = \mathrm{lcp}(x_i, x_j) = 0$, since they do not share a common prefix. Thus a Relationship Strength value of zero indicates unrelated keywords (keywords in different documents). If $\tau_i = \tau_j$ and both the keywords are directly satisfied by the same node i.e. $x_i = x_j$, then $\mathrm{RS}(k_1, k_2) = \mathrm{lcp}(x_i, x_j) = \mathrm{length}(\mathrm{hvi}(x_i)) = \mathrm{RS}_{\max}$. Thus Relationship Strength values of two keywords can take integer values in the range of $[0{:}\mathrm{RS}_{\max}]$ based on their occurrence and proximity in the XML document. The concept of Relationship Strength can be extended to a query $q = \{k_1, k_2, \ldots, k_m\}$ consisting of m terms.

For example, in the document trees in Figs. 5(a) and (b), the nodes A and B have a common prefix of length two. Thus, they have a RS value of two; similarly nodes A and C have an RS value of one. Whereas, nodes A and D have an RS value zero since they belong to different document trees.

6.3. *Semantic interconnection*

In terms of the XML document tree, two nodes are semantically interconnected if they share a common ancestor and this ancestor is not the root of the document tree. As an illustration, consider the XML document tree in Fig 4. The keywords "*Vipin*" and "1979" have a common ancestor, *inproceedings*(1). Thus, they are semantically interconnected. Whereas the keywords "*Vipin*" and "1980" have a common ancestor, *dblp*(0), which is the root of the document tree. Hence, the two keywords are not semantically connected.

Theorem 2. *Two keywords* k_1 *and* k_2 *are semantically interconnected iff* $\mathrm{RS}(k_1, k_2) > \mathrm{level}_i + 1$, *where* level_i *is the first such level in the document tree where the degree of the node is greater than one.*

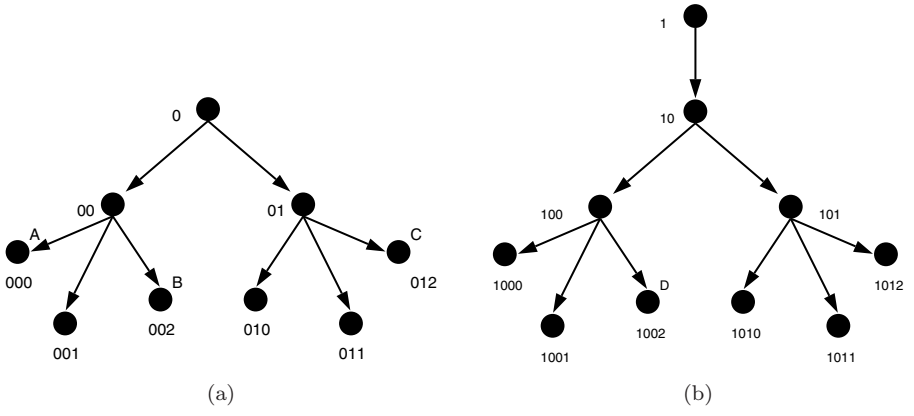

Fig. 5. Semantic interconnection.

Proof. Consider $\text{level}_i = 0$. Then the root of the document tree will have a degree greater than one. The document tree is as shown in Fig. 5(a). Since $\text{level}_i = 0$, $\text{RS}(k_1, k_2)$ should be greater than one for the two keywords to be semantically related. If $\text{RS}(k_1, k_2) > 1$, then there exist two nodes x_i, x_j that directly satisfy k_1, k_2 and with Hierarchical Vectors for Identification $\text{hvi}(x_i)$ and $\text{hvi}(x_j)$, such that $\text{lcp}(x_i, x_j) \geq 2$. Thus the two keywords have at least two common ancestors and of these only one can be the root of the document tree. Hence, the two keywords k_1, k_2 share at least one common ancestor apart from the root, and they are Semantically Interconnected. For $\text{level}_i > 0$, the document tree is as shown in Fig. 5(b) and the same proof given above holds good. □

For example, in the XML document tree in Fig. 5(a), since $\text{level}_i = 0$, RS must be greater than one for the nodes to be semantically relevant. The nodes A and B have an RS value of two and are semantically relevant. Whereas, nodes A and C have an RS value of one, and hence are not semantically relevant.

The RS can also be used to rank the semantically interconnected keywords. The semantically interconnected keywords with higher RS values are the more relevant results and hence are better ranked than those having lesser RS values. The architecture to compute semantically related results from the Most Frequently used Index (MFI) and Less Frequently used Index (LFI) is shown in Fig. 6.

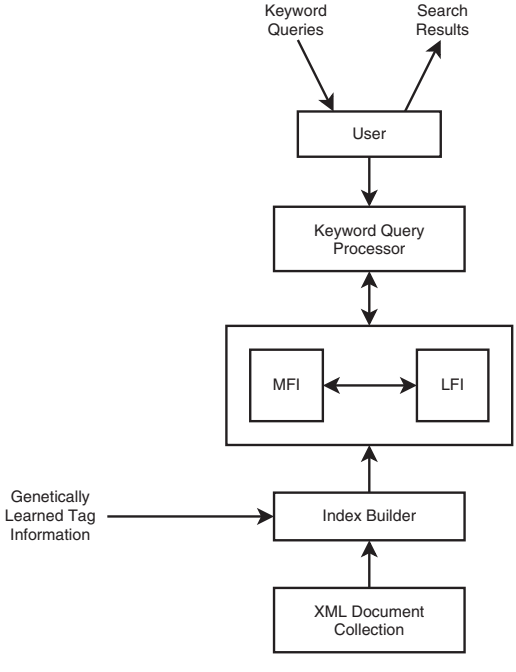

Fig. 6. Architecture of SAGAXsearch.

(1) **if** $(m = 1)$
s = search the MFI with query q
(a) **if** s = Null;
(b) s = search LFI with query q
(c) search_result = s
(2) **else if** $(m > 1)$
(2.1) **if** search with q in MFI is successful
(i) s = semantically interconnected nodes in the search results
(ii) **if** $(s$ = Null)
no semantically related nodes
(iii) **else** search_result = s
(2.2) **else**
(i) continue search with q in LFI
(ii) s = semantically interconnected nodes in the search results
(iii) **if** s = Null
no semantically related nodes
(iv) **else** search_result = s

Fig. 7. Partitioned index search.

Let $q = \{k_1, k_2, \ldots, k_m\}$ be the search query where k_i represents the ith term in the query q and m represents the total number of terms in the query. The algorithm to find the semantically interconnected elements is given in Fig. 7. The search algorithm first checks the length of the keyword query. If the query consists of a single term, a search over the MFI is performed. If the search is not successful, the algorithm continues search over the LFI. A failure to retrieve results from both MFI and LFI implies that the term is not found. The same technique is extended when searching with queries having more than one term. The only change is that, at each stage the semantic interconnection of the results is checked. Only semantically interconnected nodes are considered as the search results.

7. Experimental Results

In this section, we analyze the efficiency and accuracy of SAGAXsearch, which is implemented in Java, *via* experiments on real data. The experiments were carried out on a Pentium IV, with a CPU of 2 GHZ and 512 MB of RAM, running the Windows XP operating system.

Test Set: The test set used in SAGAXsearch is the DBLP XML corpus.[10] The DBLP XML corpus is a collection of 200,000 XML documents and is a repository for the details of a wide range of scientific articles. The documents in the DBLP corpus can be classified into two major categories: journal articles and conference papers. The structure and the elements (nodes) used to represent the two types of documents are different. The corpus makes use of 36 different elements. The

elements can be unique to a document category or they might be shared by document categories.

Genetic Learning of Tags: The Self Adaptive GA used in SAGAXsearch takes a small number of user queries (10–20 queries) and the documents adjudged as relevant by the user as inputs. The input documents to the GA are XML fragments from the DBLP XML database. The GA tries to explore all possible tag combinations from the DBLP database and tries to find the best tag combination which satisfies the maximum number of queries. The experimental result in Fig. 8 shows the average fitness for the generations of population. Note that the fluctuations in the curve representing Self Adaptive Migration model GA (SAMGA) is because of the adaptiveness introduced in the migration rate and population size. For SAMGA the average fitness steadily raises until about the fifteenth generation and then the fitness increases slowly. As the generation progresses further, the increment of fitness falls, as most of the individuals have already converged to their best fitness values. In contrast, a Simple GA (SGA) fails to converge even after 20 generations. Thus, the application of SAMGA helps in faster convergence when compared to SGA.

As the generation continues, the weights of the tags in the XML document are adjusted. The adjustments are such that maximum number of relevant results are retrieved. The DBLP XML database has a large number of distinct tags (>30). The evolution of these tag weights with the generations of the GA is shown in Fig. 9. Due to space constraints, the evolution of all the tags cannot be represented; so we illustrate the evolution in the weights of only six tags: <*author*>, <*title*>, <*year*>, <*pages*>, <*booktitle*>, *and* <*url*>. The average tag weight represents the average of the weights assigned to all the tags in the document.

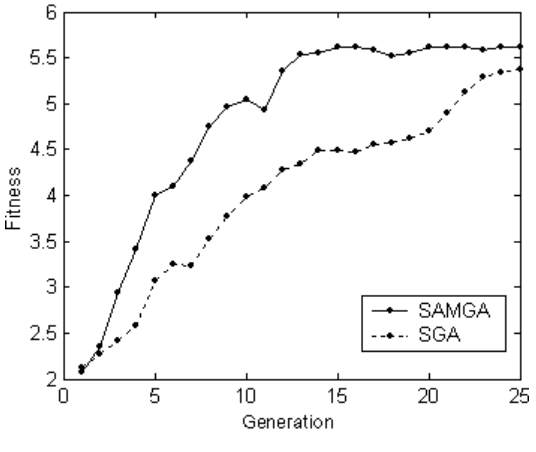

Fig. 8. Average fitness of the populations.

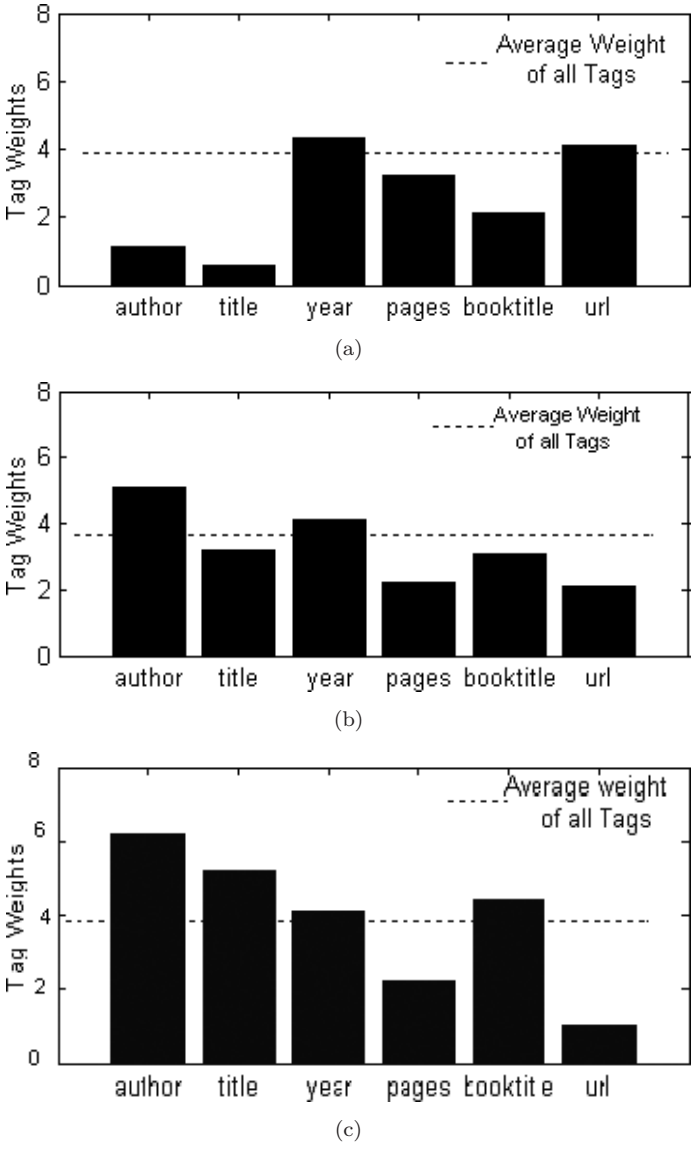

Fig. 9. (a) Tag weights at the start of GA. (b) Tag weights after fifteen generations. (c) Tag weights after termination.

The weight of a tag, when compared to the average weight of all tags in the document, is a measure of the importance of the tag within the document. Figure 9(a) shows the random tag weights assigned to the tags at the start of GA. As the generations continue, the user queries are evaluated and the tag weights are adjusted so as to obtain maximum fitness values. Figure 9(b) shows the tag weights after

Table 1. Top four tags and their corresponding weights.

Generation	Tag1 Weight	Tag2 Weight	Tag3 Weight	Tag4 Weight
1	month	author	ee	url
	7.71	6.67	6.16	4.26
5	author	pages	school	cite
	7.63	7.15	6.74	5.23
10	title	year	cite	booktitle
	7.72	6.35	5.92	5.87
15	author	year	title	pages
	8.1	7.63	7.19	6.53
20	author	year	title	pages
	8.03	7.51	6.83	6.60

fifteen generations. The tag weights after the termination of GA are the real measure of the importance of tags, and are as shown in Fig. 9(c). For the DBLP dataset, based on randomly sampled user queries, tags like <*author*>, <*title*>, <*year*>, and <*booktitle*> are classified as important. The tags like <*pages*>, <*url*>, <*cite*>, and <*ee*> failed to classify as important. Table 1 shows the tag weights of the top four tags with the largest tag weights at the end of every five generations.

8. Analysis and Comparison of Results

We now evaluate the performance of keyword queries over a subset of the DBLP XML corpus, using SAGAXsearch. Here we compare the performances of a search using a normal index and a search using MFI and LFI. A normal index is an index which stores all the XML tag information within a single flat index structure. For large XML collections, such an index becomes huge and is difficult to manage. In contrast, the MFI has an index size which is much smaller, but still is capable of satisfying a majority of the user queries. During experimentation, the normal index which we built from the subset of the DBLP XML document had a size of 51.8 MB. The same document was indexed into two separate partitions by making use of the knowledge learnt from GA. The two partitions, MFI and LFI, had sizes of 20.4 MB and 31.6 MB respectively. In addition to this, MFI was capable of satisfying about 70% of the user keyword queries and for the remaining 30% of the queries, search had to be continued with LFI. In the cases where only the MFI was sufficient to satisfy the search query the query execution time is lesser compared to XSEarch,[3] XRank.[20] This is because the index over which the search is performed is smaller. Note that the partitioned index structure that we have proposed can be used with many of the contemporary XML search techniques.

The query execution time is shown in Figs. 10 and 11. The query execution time depends upon several factors like the number of terms in the search query, the desired number of search results and the frequency of occurrence of the keywords. We experimented with variations in all these factors and found that the frequency

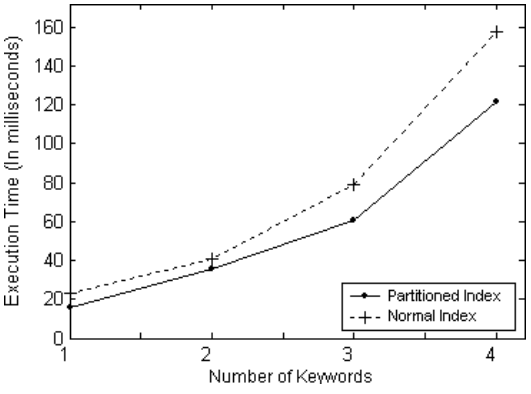

Fig. 10. Low frequency of occurrence of keywords.

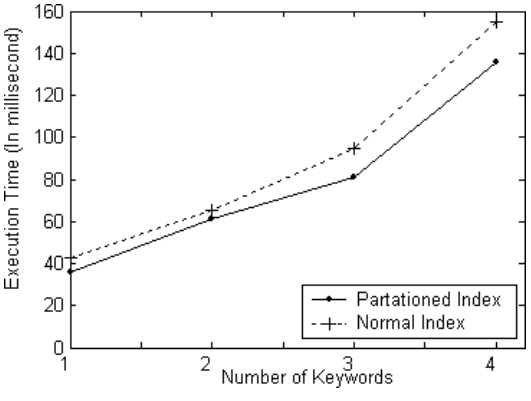

Fig. 11. High frequency of occurrence of keywords.

of occurrence of keywords was the main factor which decided the query execution time. Terms like "*database*", "*conference*", "*technique*" had very high frequency of occurrence in the DBLP document, and search queries involving these terms took longer time to execute.

Precision and Recall: Precision of the search results is the proportion of the retrieved document fragments that are relevant. Relevance is the proportion of relevant document fragments that are retrieved. Precision and recall can be represented by a contingency table as shown in Table 2.

For precision and recall, we compare the results of SAGAXsearch with XSEarch, XRank and the naïve search technique over the DBLP database. A naïve search technique is one which does not make use of the semantic information present in the XML documents. Any search technique used for information retrieval over text and HTML can be classified as a naïve technique. Such a technique is not guaranteed to

Table 2. Evaluation contingency table.

	Retrieved	Not Retrieved
Relevant	a	b
Not relevant	c	d

Recall $= a/(a+b)$, **Precision** $= a/(a+c)$.

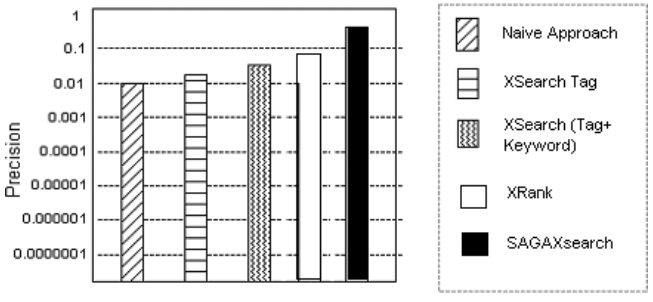

Fig. 12. Comparison of precision values.

retrieve semantically related results, and hence has low precision values. All these techniques yield perfect recall i.e. all relevant documents are retrieved, but the precision values vary. This is because of the factor that, apart from the relevant results, some irrelevant results are also retrieved. The comparison of precision values of these techniques is shown in Fig. 12.

The precision values of SAGAXsearch are found to be higher than those of XSEarch, XRank and naïve approaches, when compared over the DBLP XML dataset. The small loss in precision occurs when the same keywords are present in both the MFI and LFI and the intention of the user is to retrieve information from the LFI. In such cases, the algorithm has already retrieved the results from the MFI, it will not continue search over the LFI. The possibility of such an event is quite rare, and hence SAGAXsearch manages to exhibit high precision values.

Example: We elaborate the working of SAGAXsearch using examples. The scalability of SAGAXsearch is confirmed by the performance measures on real data as explained earlier. For simplicity and better understanding, we consider small example documents. All the example documents are that of a shopping portal and have a common structure, which is shown in Fig. 13.

Various XML documents conforming to this structure are considered as training and test sets. The training set also consists of keyword queries and the results classified as relevant by the users. During experimentation, the keyword queries are randomly sampled, from the feedback of relevance given by a large number of simulated users. Thus, the training set is not biased towards the preference of any particular user. Training queries like "*CD pack India*", "*IBM Notebook Rs 50000 Credit card*", "*cartridge for inkjet printer*", "*used car Ford Fusion*", etc. are sampled along with the relevance feedbacks of the results. The relevance feedback is

```
<item>
    <location></location>
    <price_range></price_range>
    <manufacturer></Manufacturer>
    <name></name>
    <desciption></desciption>
    <item_id></item_id>
    <payment></payment>
    <condition></condition>
    <seller></seller>
</ /item>
```

Fig. 13. Structure of example documents.

a set of XML documents which the users consider as relevant. The genetic algorithm is used to find the tag combinations which can answer the maximum number of sampled queries. The initial random weights associated with tags in the XML document are shown in Table 3.

The GA starts with this tag configuration and terminates with the tag combination satisfying the maximum number of queries. The genetically learned tag configuration which can satisfy the maximum number of queries is found to be <*price_range*>, <*manufacturer*>, <*name*>, <*description*>. Tags like <*item_id*>, <*payment*>, <*condition*>, <*location*>, <*seller*> are less frequently used during search. The tag weights after the termination of GA are shown in Table 4. With this information, two separate indices are built. MFI contains the information in the tags <*price_range*>, <*manufacturer*>, <*name*>, <*description*>. LFI contains the information in the remaining tags.

Table 3. Initial weights of tags in the example XML documents.

location	price_range	manufacturer		name
7.3	5.6	1.2		7.3
description	item_id	payment	condition	seller
2.5	4.1	4.5	6.1	1.2

Table 4. Tag weights after termination of GA.

location	price_range	manufacturer		name
1.1	6.1	7.2		7.3
description	item_id	payment	condition	seller
6.5	2.1	3.6	2.1	0.8

Consider a keyword search query like "*Motorola mobile GSM*" on the test set XML documents. The intention of the user is to find Motorola mobile phones with GSM technology. The indexing of the XML test set documents is performed offline and the hvi values for the various elements is stored in the index. When a search is performed, all elements satisfying the search terms are retrieved as results. Consider the result elements with hvi values [1 2 4], [1 2 7]. They have two prefix $\langle 1, 2 \rangle$ in common and hence have RS values greater than one. i.e. they share a common parent apart from the root. Thus, they are semantically interconnected. The search results with hvi values [1 2 4], [1 3 5] have a common prefix 1. They have an RS value of one. Such results have a single ancestor in common and hence are semantically unrelated elements. Note that the query "*Motorola mobile GSM*" can be answered by the MFI alone without the help of LFI and thus the query results are obtained faster. A modified query like "*Motorola mobile Bangalore credit card payment*" after searching over the MFI continues search with LFI, but the system has learnt that such specific queries are rarely encountered. Hence, even when such specific queries are encountered there is no loss in precision.

9. Conclusions

We have proposed a framework for information retrieval from XML documents that uses tag information to improve the retrieval performance. Genetic Algorithms, which are efficient for search in large problem spaces, are used to learn the significance of the tags. A Self Adaptive Real Coded GA is used in particular because of its ability to perform a rapid exhaustive search over a large problem space. The notations for relationship strength and semantic relationship help in efficient retrieval of semantically interconnected results as well as ranking the search results based on the proximity of the keywords. Experiment on real data show that the SAGAXsearch is accurate and efficient. It has the flexibility of keyword query search, but the results obtained maintain accuracy values comparable to that of structured queries over XML documents.

Acknowledgment

The Project is partially supported by the AICTE, as a part of Career Award for Young Teachers (AICTE File No.: F. No. 1-51/FD/CA/ (9)/2005-06) to Mr. K. G. Srinivasa, who is presently working as faculty in Department of Computer Science and Engineering, M. S. Ramaiah Institute of Technology, Bangalore — 560054, India.

References

1. S. Brin and L. Page, The anatomy of a large-scale hypertextual web search engine, *Proc. Seventh World-Wide Web Conf.* (WWW7) (1998).

2. R. Luk *et al.*, A survey of search engines for XML documents, *SIGIR Workshop XML IR* (2000).
3. S. Cohen, J. Mamou, Y. Kanza and Y. Sagiv, XSEarch: A semantic search engine for XML, *VLDB 2003*, pp. 45–56.
4. M. Srinivas and L. M. Patnaik, Genetic algorithms: A survey, *IEEE Comput.* **27**(6) (1994) 17–24.
5. K. G. Srinivasa, K. Sridharan, P. D. Shenoy, K. R. Venugopal and L. M. Patnaik, A dynamic migration model for self-adaptive genetic algorithm, *Proc. Intell. Data Eng. Automated Learning-IDEAL 2005*, Brisbane, Australia (2005), pp. 555–562.
6. W. M. Spears and K. A. De Jong, Using genetic algorithms for supervised concept learning, *Proc. 2nd Int. IEEE Conf. Tools Artif. Intell.*, Herndon, Virginia, USA (1990), pp. 335–341.
7. W. N. Martin, J. Lienig and J. P. Cohoon, Population structures — Island (migration) models: Evolutionary algorithms based on punctuated equilibiria, in *Handbook of Evolutionary Computation*, eds. T. Black, D. B. Fugee and Z. Michalewicz (Institute of Physics Publishing and Oxford University Press, 1997), pp. C6.3:1–C6.3:16.
8. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt and J. F. Naughton, Relational databases for querying XML documents: Limitations and opportunities, *VLDB* (1999), pp. 302–314.
9. S. Abiteboul, On views and XML, *SIGMOD Rec.* **28**(4) (1999) 30–38.
10. DBLP XML Records http://acm.org/sigmoid/dblp/db/index.html (February 2001).
11. J. Shanmugasundaram *et al.*, A general technique for querying XML documents using a relational database system, *SIGMOD Rec.* **30**(3) (2001) 20–26.
12. World Wide Web Consortium XQUERY: A Query Language for XML W3c Working Draft, http://www.w3.org/XML/Query.
13. D. Florescu, D. Kossmann and I. Manolescu, Integrating keyword search into XML query processing, *Int. J. Comput. Telecommun. Networking* **33**(1) (2000) 119–135.
14. M. Gordon, Probabilistic and genetic algorithms for document retrieval, *Commun. ACM* **31** (1988) 1208–1218.
15. J. Yang and R. R. Korfhage, Effects of query term weights modification in annual document retrieval: A study based on a genetic algorithm, *Proc. Second Symp. Doc. Anal. Inf. Retrieval* (1993), pp. 271–185.
16. J. Yang, R. R. Korfhage and E. Rasmussen, Query improvement in information retrieval using genetic algorithms: A report on the experiments of the TREC project, *Proc. First Text Retrieval Conf.* (TREC-1) (1993), pp. 31–58.
17. P. Pathak, M. Gordon and W. Fan, Effective information retrieval using genetic algorithms based matching functions adaptation, *Proc. 33rd Hawaii Int. Conf. Syst. Sci.* (2000).
18. S. Kim and B. T. Zhang, Genetic mining of HTML structures for effective Web document retrieval, *Appl. Intell.* **18** (2003) 243–256.
19. V. Hristidis, Y. Papakonstantinou and A. Balmin, Keyword proximity search on XML graphs, *Int. Conf. Data Eng.* (2003).
20. L. Guo *et al.*, XRANK: Ranked keyword search over XML documents, *ACM SIGMOD* (2003).