9-1999

WEDAGEN: A synthetic web database generator

Pallavi PRIYARDARSHINI

Fengqiong QIN

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Wee-Keong NG

DOI: https://doi.org/10.1109/DEXA.1999.795277

WEDAGEN - A Synthetic Web Database Generator

Priyadarshini Pallavi Fengqiong Qin Ee-Peng Lim Wee-Keong Ng

Centre for Advanced Information Systems, School of Applied Science
Nanyang Technological University, Singapore 639798, SINGAPORE

{sa0169973,p144295033,aseplim,awkng}@ntu.edu.sg

Abstract

To improve searching and processing of information on the web, a web warehousing system called WHOWEDA is being developed at the Centre for Advanced Information Systems (CAIS). This system incorporates a *Web Information Coupling Model* that describes the web objects and their relationships and a *web algebra* consisting of web operators to manipulate the web objects. In order to measure the performance of WHOWEDA and similar systems that manipulate web information, a synthetic web database generator called WEDAGEN (WEb DAtabase GENerator) has been developed. It has the capability of generating web databases of different sizes and complexities determined by a set of user specified parameters. This paper presents the issues in the design and implementation of WEDAGEN. It also gives a detailed description of its system components and the strategy to generate synthetic web databases. A preliminary assessment of the use of WEDAGEN has been reported.

1 Introduction

1.1 Background

The Wold Wide Web (WWW) is fast becoming an invaluable repository of information. It can be viewed as a huge collection of graphs with nodes representing web pages and links representing hyperlinks among these pages. Almost all information available on the web can be reached through the hyperlinks in the hypertexts, but searching web information is a tedious process owing to the unorganised and heterogeneous nature of web information and the amount of network overheads involved in fetching web pages from remote sites.

To make information retrieval possible and convenient for huge amount of WWW information, a number of search engines and browsers have been developed. Despite the efforts, these search mechanisms suffer from some serious shortcomings below:

- The time delay in manually navigating the web pages and their hyperlinks is so high that the required web information cannot be obtained in a timely manner.
- The search results from the available search engines are often overwhelming and a lot of unwanted information is returned.

- The existing retrieval tools do not permit web information harnessed over a period of time to be organised and stored in some persistent storage so that it can be further manipulated or used for data mining.

In other words, the web browsers and the search engines are not the best ways to systematically harness information from the web.

At the Centre for Advanced Information Systems (CAIS), a Web Warehousing System [3] is being developed to store and manipulate web information. The system named WHOWEDA (WareHouse Of WEb DAta) stores extracted web information as web tables and provides several web operators eg. web join, web select, global coupling, etc., to manipulate web tables [4, 5, 3, 2].

During the implementation of WHOWEDA, it is necessary to perform systematic testing on the system and to evaluate its system performance. While it is possible for WHOWEDA to be tested or evaluated using actual web pages downloaded from WWW, the amount of time required for such testing and evaluation would be so high that a comprehensive experimentation would not be possible. To overcome these difficulties, we have proposed to develop a synthetic web database generator called WEDAGEN to efficiently generate synthetic web information for performance evaluation and testing purposes.

To our best knowledge, WEDAGEN is the first synthetic web database generator developed. It can also be used to generated a collection of web pages. In some way, a collection of web pages may look similar to the objects in an object-oriented database (OODB), and there are a few performance benchmarks designed for OODBs, e.g., 001[8], 007[7], Hypermodel[1] and Sequoia 2000 [9]. Nevertheless, these benchmarks do not have have a synthetic database generator that produces databases with different web schema complexities and sizes.

1.2 Objectives

In the following, we present the major objectives to be accomplished by the proposed synthetic web database generator:

- WEDAGEN is designed to create a testbed for conducting experiments to evaluate the performance of web database systems such as WHOWEDA. A generator such as WEDAGEN can also be used to test the functionalities of WHOWEDA. WHOWEDA is currently in the development phase. Although the idea of having WEADGEN is conceived as a part of the WHOWEDA project, WEDAGEN is also very useful for any other similar systems that manipulate web information. Apart from generating web information in the form of web tables, WEDAGEN is expected to generate actual web pages with calibrated content and links. Such synthetic web pages can be readily used by any web database system for performance evaluation or system testing.

- The process of retrieving information from the web can be very time consuming, especially if a very large amount of web data is required. A slow internet connection or a failed remote server can pose serious problems to a web database query processor. If web information is synthetically generated, there is no longer any dependency on the remote sites. WEDAGEN is expected to overcome the limitations of using real web information by having efficient algorithm to generate web information.

- If web data is retrieved directly from the internet, the result may become so overwhelming that it is impossible to control the quantity and quality of the information returned. This causes problems in systematic testing and performance evaluation of a web database system. For example, it may not be easy to control the number of web pages that are linked together in a required connectivity configuration, and the web pages that satisfy certain search criteria. WEDAGEN attempts to give some flexibility and control to the web database system developers to generate the appropriate web information for their uses, by allowing them to specify input parameters and input web schemas. The web tables and the web pages are therefore synthesized according to the constraints specified by the input schema and parameters. This makes the synthetic web database more appropriate for performance evaluation and system testing.

1.3 Paper Outline

The rest of this paper is structured as follows: Section 2 gives an overview of WHOWEDA, its basic concepts and the storage mechanisms. It is designed to provide a better understanding of the synthetic web database generation process. Section 3 discusses the design and an overall web information generation strategy adopted by WEDAGEN. Section 4 deals mainly with the input parameters required by WEDAGEN. Sections 5 and 6 describe the creation of node and link instances and the generation of web tuples, respectively, as the two important steps in synthetic web table generation. Section 7 gives a preliminary assessment of WEDAGEN. Finally, conclusion and the future work are presented in Section 8.

2 WHOWEDA - An Overview

To realize a web warehousing system, a data model called the Web Information Coupling Model(WICM) was introduced to describe and abstract web objects [3] . The proposed data model consists of a collection of web objects known as **nodes** and **links**. Nodes are essentially web pages. Node objects are characterised by a set of attributes namely *URL, title, format, size, date and text.* The *format* of a node (web page) refers to the file type of the node which can be HTML, postscript, plain text etc.. Links are the hyperlinks interconnecting the web pages. The link objects have *source_URL, target_URL, label and link_type* as the main attributes. The *link_type* of a link can be interior, local or global. Based on the web objects, one derives the web tuples and web tables. A **web tuple** is a set of connected and directed graphs consisting of nodes and links. A collection of web tuples described by a web schema is called a **web table**.

A **web schema** describes the meta-information that binds the tuples in a web table. The meta information consists of a set of node variables, link variables, connectivities and predicates defined on attributes of the node and link variables. Hence, a web schema can be represented as a directed graph. A *source node* in a schema refers to the node which has no incoming links to it i.e., it is the start node of the schema graph. The same schema may have more than one source node. A *sink node* is one which has no outgoing link.

Figure 1 depicts an example schema of a web table stored as a schema file. Figure 2 shows the same schema graphically.

In the schema shown:

- a, b, and c are the node variables. Node a is the source node while c is the sink node.
- e and f are the link variables.

```
[NODES]
a
    URL EQUALS “http://www.ntu.edu.sg”
b
    title CONTAINS “commerce”
c
    title CONTAINS “market”

[LINKS]
e
    label EQUALS “research”
f
    label CONTAINS “group”

[CONNECTIVITIES]
a⟨e⟩b
b⟨f⟩c
```

Figure 1: An Example Web Schema.

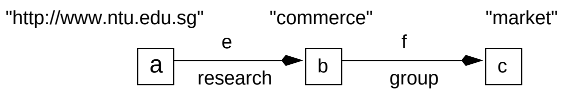

Figure 2: Graphical Representation of Schema.

- inter-connectivities among the node variables have been defined. For example, node variable a is linked to node variable b via the link variable e.
- predicates have been defined on the nodes and link variables. For example, Title CONTAINS “commerce” is a predicate defined on the node variable b. The predicate requires instances of b to have “commerce” included in their titles. A predicate on any text-based attribute of a node or link variable can involve either *CONTAINS* or *EQUALS*. The argument of *CONTAINS* predicate is a list of query words while that of *EQUALS* is a string. In the example given,

To distinguish between individual node and link instances, unique IDs are assigned to them. A node ID uniquely identifies a node instance. Similarly, all link instances are assigned unique link IDs.

The WHOWEDA system can maintain multiple web databases each consisting of a set of web tables[3]. While the web tables may be distinct in their content, they may share common web pages associated with their node instances. In WHOWEDA, a web table is therefore stored as a set of files described below:

- *Table files:* This refers to the web schema and web tuple files storing the schema and tuples of a web table respectively. Each tuple in the web tuple file consist of node and link instances forming the tuple. The web tuple file maintains the node attributes like the node ID and *URL*, and link attributes like *target node ID*, *label* and *link_type* for those node and link instances involved in the tuples.

- *Node pool file:* Since a node instance may be involved in multiple web tuples, the complete meta-information about the nodes are stored in a single node pool file shared by all web tables. Meta-information about a node instance includes the node ID, *URL*, name of web page corresponding to the node instance, *format*, *size*, *text* and complete information of its outgoing links.

- *Web pages:* This are files storing the actual web pages generated for the individual node instances.

3 The Design of WEDAGEN

3.1 Design Objectives

WEDAGEN is to be used for performance evaluation and system testing of a web database system. In the design of WEDAGEN, several criteria have been considered. The implication of these design criteria and the decisions we made in the design of WEDAGEN are described below:

- *Scalability* - To meet the objectives given in Section 1.2, the synthetic database generator should be designed to generate variable amount of data. We would like to be able to generate databases consisting of web documents in the order of millions. The database size should be large enough to be realistic for performance evaluation as query performance differenes are magnified on large databases. In WEDAGEN, scalability is achieved by allowing users to specify input parameters that determine the size of the database to be generated and ensuring that the generator algorithm is able to produce as much as can be accommodated by the underlying operating system.

- *Flexibility* - The flexibility criteria refers to control over the structure of web information to be generated. In the case of WHOWEDA, the structure of web information refers to schemas of the web tables. For testing and performance evaluation of web database systems, it is necessary to have tables with varying complexities. WEDAGEN is designed to accept a web schema as an input parameter, and web schemas of different topologies and predicates can be specified.

- *Simplicity* - As web schema becomes an input parameter of WEDAGEN, users may have to specify a separate set of parameters for each node and link variables in the web schema. The parameter specification process could therefore be painstaking and error-prone. To ensure that WEDAGEN is easy to use, we have designed WEDAGEN to support default parameter values to be assigned to node and link variables if specialized parameter values are not required.

3.2 System Overview of WEDAGEN

Figure 3 depicts the overall architecture of WEDAGEN. It consists of three modules namely, *specific parameter generation module*, *instance generation module* and *tuple extraction module.*

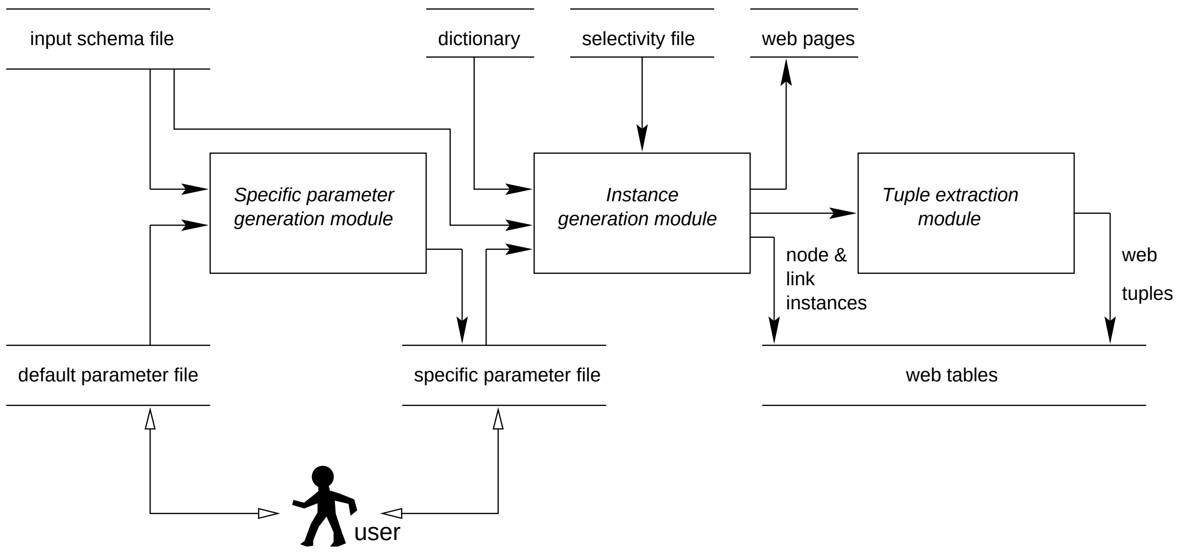

Figure 3: System Architecture of WEDAGEN.

As described in Section 2, every web table has a schema consisting of node variables, link variables, connectivities, and predicates. The web schema has to be given to WEDAGEN as an input parameter. To control the size of a web table to be generated and the connectivities between its nodes instances, WEDAGEN requires other input parameters to be given by the users. Many of these input parameters are associated with the node and link variables in the schema. Instead of having a user specifying the same set of input parameters for each and every node and link variables, WEDAGEN allows default parameter values to be assigned to these variables. This set of default parameter values is stored in the *default parameter file.*

Given the input schema and the default parameters, the *specific parameter generation module* generates a *specific parameter file* containing the default parameter values assigned to all the node and link variables in the schema. Within the specific parameter file, the user can modify the parameter values for individual node and link variables to control the actual composition of node and link instances finally generated.

Once the input parameters for individual node and link variables are available, the *instance generation module* will generate the node and link instances for the web table according to the parameter values in specific paramter file. A set of selectivity parameters can be given to constrain the proportion of web pages satisfying some predicates not found in the web schema. These are specified in the *selectivity parameter file.* The instance generation module also generates the actual web pages for the node instances. In WEDAGEN, all the words required to form the *URLs*, *titles* and *text* of web pages are obtained from a dictionary containing a huge collection of words. (See Section 5 for more details)

The last module of WEDAGEN, i.e., the *tuple extraction module*, constructs tuples for the web table from the node and link instances generated by the instance generation module. Each tuple is a list of node IDs and link IDs which satisfy the input web schema. The tuples extracted are stored in the web table files. The web table files are now complete with the node, link and tuple information. For web database systems not using web table constructs, the web pages generated can be directly used for evaluation or testing purposes.

4 Input Parameters for WEDAGEN

In this section, we give a detailed description of all input parameters required by WEDAGEN. To synthesize a web table consisting of a collection of interlinked web documents for experimentation, it is necessary to exercise some control over the size, complexity and content of the web table generated through the following four categories of input parameters.

- *Web schema parameter*: This refers to the schema of the web table to be generated
- *Instance parameters*: This refers to the parameters associated with the given web schema and are used for data generation.
- *Control parameters*: This refers to parameters associated with the node and link variables of the web schema for data checking purposes.
- *Selectivity parameters*: This refers to parameters that impose constraints on the text content of the web pages to be generated in additional to those specified in the web schema.

Note that both the instance and control parameters are included by the default and specific parameter files. Most of the instance and control parameters are associated with node and link variables. Hence, they are the input parameters that have to be specified for each node and link variable in the specific parameter file.

In the following subsections, these above input parameters are described in more detail.

4.1 Web Schema Parameter

Since the aim of WEDAGEN is to support the evaluation of web database system performance and the testing of its functionalities, it is desired for WEDAGEN to generate web tables with different schemas. The schema parameter consists of four components, namely nodes, links, connectivities and the predicates specified in the format illustrated by Figure 1. At present, WEDAGEN supports only schemas containing no cyclic connectivities among their node variables. The web schema parameter can be either specified using a text editor or automatically generated using a schema generator.

4.2 Instance Parameters

This set of parameters determines the properties of individual node and link instances to be generated by WEDAGEN. The parameters are directly used to generate the node and link instances, and to construct web tuples.

- **Approximate number of tuples (*NumTuples*):**
  This parameter indicates the approximate number of tuples in the web table to be generated. Since *NumTuples* is merely used as a guideline to determine final number of tuples generated, the actual number of tuples created may differ.
- **Number of instances for a source node variable (*NumSourceNodeInstances*):**
  This parameter determines the number of node instances to be generated for a source node variable. The node instances generated must satisfy the predicate(s) defined on the source node variable. It is not necessary to specify a similar parameter for each non-source node

variable since the number of non-source node instances can be computed directly or indirectly from the number of source node instances by considering the fanout from each node instance (See Section 5).

For example, let a be a source node variable with the predicate a.title CONTAINS "java", and a *NumSourceNodeInstances* of 5. WEDAGEN will generate 5 node instances for a and these instances have their titles containing "java". Note that if a predicate of the form a.URL EQUALS "http://xxx.yyy.com" is specified on a source node variable, the *NumSourceNodeInstances* value will not be used since only one a node instance should be generated.

- **Fan-out per node instance for a link variable (*FanOut*):**
  This parameter controls the number of outgoing links from a node instance to other node instances through a specific link variable. For example, given a connectivity $a\langle e\rangle b$ with predicate e.label CONTAINS "text", and a fanout of 7 specified on the link variable e, WEDAGEN will generate 7 outgoing links from each a instance to the b instances. Each link will be associated with a label containing the word "text".

- **Approximate number of keywords per node instance for a node variable (*NumKeyWordsPerNodeInstance*):**
  This parameter determines the number of keywords to be included by the *text* attribute of each node instance of a node variable. Furthermore, these keywords are indexed for efficient keyword searching. The keywords to be included are randomly chosen from a dictionary containing about 30,000 words retrieved from the WWW. The keywords found in the text attribute of a node instance must also be included in the web page associated with the node instance. Note that if a predicate of the form b.text CONTAINS "java" is specified on a node variable b as part of the schema, the query word java will also be included in the text attribute of each b instance. Since *NumKeyWordsPerNodeInstance* does not include such query words, it only determines an approximate number of keywords finally generated.

- **Approximate number of words per node instance for a node variable (*NumWordsPerNodeInstance*):**
  This parameter controls the size of the web page generated for each node instance of a node variable. The words in the generated web pages are again chosen from the dictionary. The parameter value excludes the query words found in the predicates associated with the node variable, but includes the number of keywords associated with the text attribute of the node variable. Hence, *NumWordsPerNodeInstance* of a node variable is always larger than or equal to the corresponding *NumKeyWordsPerNodeInstance*. For example, let b.text CONTAINS "school" be a predicate on the node variable b in the web schema, and the values of *NumKeyWordsPerNodeInstance* and *NumWordsPerNodeInstance* be 3 and 10 respectively. The word "school", the 3 keywords that form the text attribute, and 7 other words randomly chosen from the dictionary will form the text of the web page of each b node instance.

- **Approximate number of words for each link label (*NumWordsPerLinkLabel*):**
  This parameter controls the length of the label associated with each link variable in the input schema. The words used in the link label are randomly chosen from the dictionary. The query word(s) in the predicate defined on the link label of a link variable will also be included by the link label of a corresponding link instance. For example, let *NumWordsPerLinkLabel* be 10 for link variable e that connects node variables a and b, i.e. $a\langle e\rangle b$. Ten randomly chosen

words and the query word(s) appearing in the predicate(s) on e will be included in the link label associated with each e instance. Note that when a predicate of the form e.label EQUALS "java" is given in the schema, each link instance corresponding to e will only contain the word java, i.e. *NumWordsPerLinkLabel* does not apply to e.

- **Approximate number of words for the hostname of the URL assigned to each node instance of a node variable (*NumWordsPerHostname*):**
  This parameter controls the number of words used to construct the hostnames of the URLs generated for the node instances of a node variable. The *NumWordsPerHostname* of a URL is defined by the number of words separated by dots and it excludes the default prefix word "http://www.". For example, a *NumWordsPerHostname* of 3 will lead to URL of the form http://www.word1.word2.word3/ where word1, word2, and word3 are randomly chosen from the dictionary.

- **Approximate number of words for the web page title of each node instance of a node variable (*NumWordsPerTitle*):**
  This parameter controls the number of words to be chosen to form the title of a web page corresponding to a node instance of a node variable. Similar to *NumWordsPerNodeInstance*, *NumWordsPerTitle* does not include the query word(s) of a predicate specified on the title of the node variable.

- **Type of link for a link variable (*LocalGlobalLink*):**
  This parameter determines if local or global links are to be generated for a link variable. When the link type of a link variable is global, i.e. $LocalGlobalLink = 0$, the URLs of the source and destination node instances that are associated with the link variable will carry different site addresses. A local link type is indicated by having $LocalGlobalLink = 1$.

All instance parameters except *NumTuples* are associated with some node or link variable. Hence, these instance parameter values in the default parameter file are duplicated for every node and link variables in the specific parameter file by the specific parameter generation module.

4.3 Control Parameter

At present, only one control parameter is used in WEDAGEN. Unlike the other parameters, a control parameter is not used for data generation. It indicates some criteria to be satisfied by the node and link instances generated. In order to satisfy the criteria, WEDAGEN may have to require the user to modify the instance parameters and re-generate the synthetic node and link instances.

- **Maximum Fan-in of each node instance of a node variable (*FanIn*):** To avoid clustering of incoming links to a particular node instance, the maximum fan-in of the node instances can be specified.

Like most instance parameters, *FanIn* parameters are associated with node variables. Hence, the parameter is found in the default parameter file, and is duplicated for all node variables in the specific parameter file. A user can modify the *FanIn* parameter for each node variable to impose different fan-in criteria for different node variables.

After all node and link instances have been generated by the instance generation module, they are checked against the *FanIn* parameters associated with the node variables. If any of the node instances has a fan-in exceeding the *FanIn* parameter for its node variable, WEDAGEN will alert the user who may choose to modify the default parameter values and re-generate the node and link instances.

4.4 Selectivity Parameters

The selectivity of a predicate refers to the percentage of the node instances that satisfy the given predicate. While WEDAGEN always ensures that all predicates in the given web schema will be satisfied by the node and link instances generated, it is still useful to impose some additional constraints on the text content of the web pages corresponding to the generated node and link instances. In particular, the specially calibrated content of web pages can be used to evaluate the performance of web operations and web queries. In WEDAGEN, the constraints on the text content of the web pages to be generated are determined by a set of predicates together with their selectivities. Note that two kinds of predicates can be specified, namely predicates associated with some node variables, and predicates associated with the entire web table.

- **Selectivity of a predicate specified on some node variable (*NodeSelectivity*)**:
  This parameter determines the number of instances of a particular node variable satisfying a predicate not found in the input web schema. For example, given a predicate a.URL CONTAINS "language" and its selectivity of 0.6, WEDAGEN will generate 60% of the node instances of a having their URLs containing language.

- **Selectivity of a predicate specified on the entire web table (*TableSelectivity*)**:
  A predicate specified on the entire web table is one that does not require any node variable. Given such a predicate and its selectivity, WEDAGEN will generate the corresponding proportion of node instances in the entire web table (irrespective of their node variables) satisfying the given predicate. For example, the predicate title CONTAINS "hello" with selectivity 0.4 will lead to 40% of the node instances having their titles containing "hello".

NodeSelectivity and *TableSelectivity* are specified in the selectivity parameter file. Each parameter entry consists of a predicate and its selectivity value.

5 Instance Generation Module

This section describes in detail the *instance generation module* which creates the actual node and link instances and stores them in the node pool file and in the web table files. The major steps in instance generation are summarized as follows:

- **Step 1: Determining the number of instances for each non-source node variable**
  In this step, we calculate the number of instances for each non-source node variable since this piece of information is not directly provided by the user. The number of node instances of a non-source node variable is computed from the number of node instances of the other node variables having direct links to it and the fanouts associated with these links.

  For example, suppose c is a non-source node variable involved in two connectivities, $a\langle e\rangle c$ and $b\langle f\rangle c$ where a and b are source node variables. Let the *NumSourceNodeInstances* values of a and b be 5 and 4 respectively, and the *FanOut* of e and f be 2 and 3 respectively. The number of c instances can be computed by maximum of (*NumSourceNodeInstances of* $a\times$ *FanOut of* *e*) and (*NumSourceNodeInstances of b*× *FanOut of f*). That is, there are 12 *c* instances to be generated.

Note that if a predicate of the form *c*.URL EQUALS "http://www.xxx.yyy/zzz" is associated with the non-source node variable *c*, there will be only one *c* instance to be generated.

Based on the above strategy to determine the number of node instances for node variables, one can derive the following lemma:

Lemma 1 *Given a schema with no predicate of the form* ⟨*node_var*⟩.URL EQUALS ⟨*URL_String*⟩, *the number of node instances for a node variable x is always no larger than that of any node variable to which x is linked.* ∎

Due to Lemma 1, one can conclude that the number of instances to be generated for a sink node variable is no fewer than that of its predecessor node variables.

- **Step 2: Generating the URLs of node instances**
  Having determined the number of instances for each node variable, the URLs of all node instances are generated. Recall that predicates on URLs may be specified within a web schema. The instance generation module would ensure that the URLs generated satisfy these predicates. Each URL consists of http followed by a hostname and a filename for the web page determined by the node variable name and node id of the corresponding node instance. The length of hostname in a URL is determined by the *NumWordsPerHostname* parameter. Moreover, the proportion of URLs generated must also satisfy the selectivity parameters involving URLs.

- **Step 3: Creating node and link instances**
  This step determines the values of node attributes such as *text*, *date* etc., creates the outgoing link sets of node instances, and constructs the corresponding web pages. All generated node and link instances are stored in the node pool file and web table files. The various sub-steps have been elaborated below:

  - **Creating non-URL attributes of node instances:** To create the *text* attribute of a node instance, *NumKeyWordsPerNodeInstance* words are randomly chosen from the dictionary. The *text* attribute of the node instance may also include some words suggested by predicates in the schema and the selectivity parameters. All keywords in the *text* would also be part of the textual content of the corresponding web page. Similar to the *text* attribute, the instance generation module generates the *title* attribute based on instance parameters, web schema, and selectivity parameters. At present, the *dates* of the node instances are simply the current date.
  - **Generating link sets for node instances:** Having generated the attributes of node instances, the outgoing links of each node instance have to be created before the instance generation module proceeds to create the web pages. The number of outgoing links from a particlar node instance is generated based on the *FanOut* parameter associated with the corresponding node variable. The target URL of each link instance is determined randomly from the URLs of node instances corresponding to the target node variable. To create the link label of a link instance, *NumWordsPerLinkLabel* words are randomly selected from the dictionary under normal circumstances. When the input web schema

includes some predicate(s) on the link label of a link variable, the instance generation module must ensure that the the link labels of the corresponding link instances satisfy the predicate.

– **Creating web pages:** The node and link information generated so far are written into web pages. At the same time, the actual textual content of these web pages are randomly generated using the *NumWordsPerNodeInstance* parameters of the node variables. The instance generation module also includes in the generated web pages some images randomly chosen from a collection of GIF files. The size of the node instance is then determined by the size of its web page.

In Step 1, we determine the number of instances of non-source node variables including the sink node variables. We define *maximum number of sink node instances* of a schema to be the number of sink node instance for the sink node variable that is assigned the max number of instances based on the computation in step 1.

Recall that a web schema is a graph consisting of node and link variables. We are going to show that when a new schema is derived by adding a link or reversing a link of an existing schema graph, the maximum number of sink node instances will increase.

To define the above property, we first give the definitions of *insert-link* and *reverse-link* operations:

Definition 1 Insert-Link *operation adds a new link between two node variables in a schema graph. The link is defined by a link variable and a fan-out value associated with it.* ∎

Definition 2 Reverse-Link *operation reverses the direction of a link between two node variables such that the number of instances computed for the original destination node variable, and the fanout out assigned to the original link are unaffected.* ∎

After a insert-link or reverse-link operation, the number of instances for each node variable reachable from the new link has to be recomputed based on the Step 1 of instance generation.

Theorem 1 *Let G_1 be a directed, acyclic schema graph and G_2 be another schema derived from G_1 by a sequence of insert-link and reverse-link operations.*
Maximum number of sink node instances of G_2 $\geq$ Maximum number of sink node instances of G_1

Proof
Let m be maximum number of node instances for sink node variable in G_1. Let op_1, op_2.......op_k be the sequence of k operations performed on G_1 in order to derive G_2. For each operation op_i, there can be two possible cases:

- *Case 1*: op_i is an *insert-link* operation involving $node_i$ and $node_j$.
  Let number of instances of $node_j$ be n_j in G_1 and the new link edge is inserted such that $node_i$ is the source node and $node_j$ is the target node. After performing op_i, number of instances of $node_j$ will be the maximum of (n_j, *number of instances of* $node_i$ $\times$ *Fanout*). This implies that the new number of instances of $node_j$ after performing the operation would be $\geq n_j$. If $node_j$ is not the sink node, as the number of instances of $node_j$ increases, the number of instances of its successor nodes (including the sink node) will also increase. To illustrate the insert-link operation, consider the schema *Graph1* in Figure 4. If values of *NumSourceNodeInstances* and

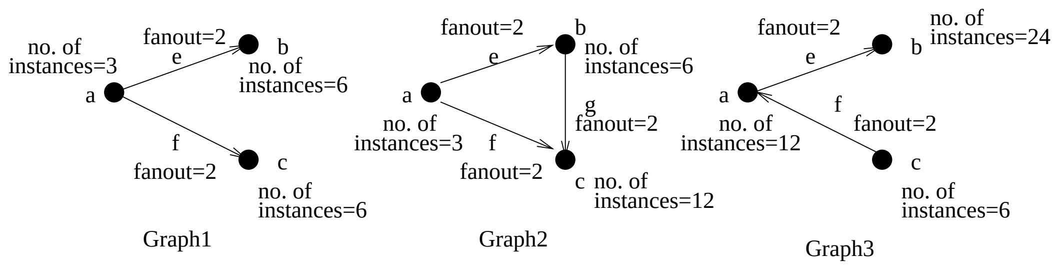

Figure 4: Schema graphs depicting insert-link and reverse-link operations.

FanOut are 3 and 2 respectively, then the number of instances of b and c are 6 each (=2×3). A new link g having a *FanOut* of 2 is added with b as the source node and c as the sink node as a result of the insert-link operation (see Graph2 in Figure 4). The number of instances of c is now the maximum of (6, 12), i.e. 12. Hence, the maximum number of sink node instances increases after insert-link operation is performed.

- *Case 2*: op_i is a *reverse-link* operation involving $node_i$ and $node_j$.
  Let the original schema contains a link from $node_i$ to $node_j$, and the reverse-link operation reverses the direction of the link. Since reverse-link operation does not change the number of instances for $node_j$, only the numbers of instances for those node variables reachable by $node_j$ need to be recomputed. Note that if $node_j$ itself is originally a sink node variable, the maximum number of sink node instances of the graph can only increase due to Lemma 1 as the reverse-link operation does not remove other sink node variables. If $node_j$ is not a sink node variable originally, the maximum number of sink node instances may also increase due to the possibility of having new sink node variables after the reverse-link operation.

  For example, consider *Graph1* of Figure 4. Assuming *NumSourceNodeInstances* and *FanOut* values are 2 and 3 in *Graph1*, the maximum number of sink node instances is 6. *Graph3* is derived from *Graph1* by reversing the direction of link *f*. Node c which was a sink node in *Graph1* becomes a source node in *Graph3* with *NumSourceNodeInstances* equal to that of the number of instances computed for the original c in *Graph1*. After the operation, new maximum number of sink node instances i.e. number of instances of b, becomes 24.

The theorem is true for individual operations discussed above. So, by *induction*, the condition is true for all combinations of the insert-link and reverse-link operations. Hence, the theorem stated above is proved. ∎

6 Tuple Extraction Module

The instance generation module generates only node and link instances inter-connected as directed graphs but not the individual tuples. It is therefore the responsibility of tuple extraction module to extract tuples from the node and link instances by navigating the directed graphs composed of these instances.

Figure 5 depicts an example directed graph of node and link instances for the input web schema involving node variables a, b, c and d, and link variables e and f, and connectivities $a\langle e\rangle b$ and $a\langle f\rangle c$. The IDs of the node and link instance are given in Figure 5. The tuples extracted by tuple

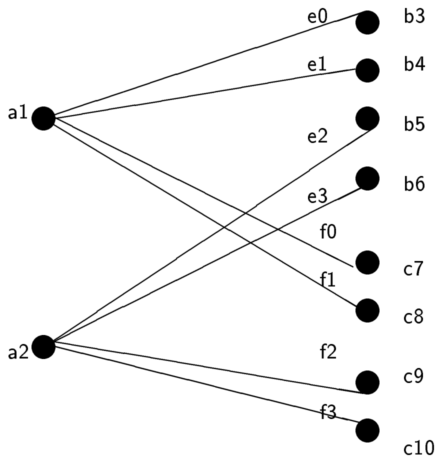

Figure 5: Node and Link Instances From Which Tuples are Extracted.

extraction module are shown in Figure 6. Each tuple is represented by the IDs of the node and link instances involved in the tuple. The tuples generated are stored in the web table files. Hence, the web table files are now complete with all node, link and tuple information.

7 Preliminary Evaluation of WEDAGEN

WEDAGEN has been implemented in C++ using Standard Template Library (STL) [6] on a SUN Ultra Workstation. Users can choose to invoke either the specific parameter generation module alone, or the instance generation and tuple extraction modules, or all three. Hence, WEDAGEN can be executed in three possible ways:

- To invoke only the specific parameter generation module, the following command is used:

  ```
  $ WEDAGEN -spgen parameter_file.def -s schema_file.shm -o parameter_file.spf
  ```

  In the above command, the flag `-spgen` indicates that only the specific parameter generation module should be invoked. The default parameter file and input schema file have the file extensions `.def` and `.shm` respectively. The specific parameter file created can be assigned a filename using the flag `-o` and it has a file extension of `.spf`.

- To invoke only the instance generation and the tuple extraction modules, the following command is used:

  ```
  $ WEDAGEN parameter_filename.spf -s schema_file.shm [-u selectivity_file]
  [-i image_file]
  ```

  Other than the specific parameter file and schema file, the other operands in the above command are optional. Note that `image_file` stores the names of the GIF files that can be included in the generated web pages.

- To directly synthesize a web table using all the default parameter values (i.e., to invoke all the three modules together), the following command is used:

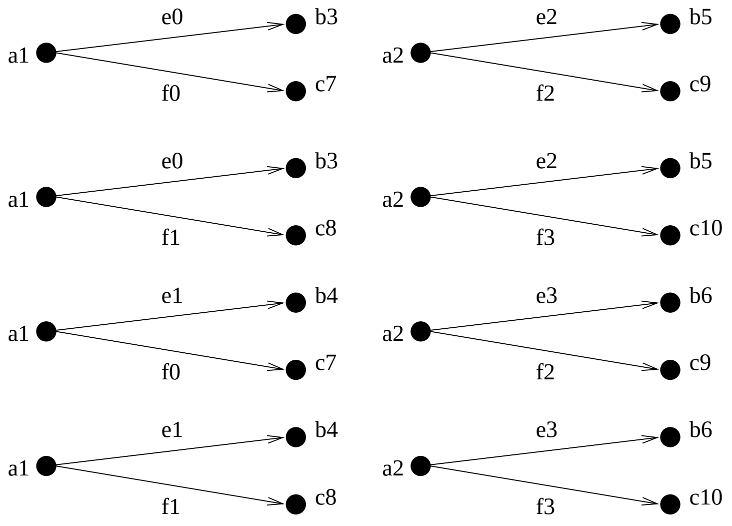

Figure 6: Extracted Tuples.

```
$ WEDAGEN parameter_filename.def -s schema_file.shm [-u selectivity_file]
[-i image_file]
```

In this section, we describe a preliminary evaluation of the web database generation capabilites of WEDAGEN. Our evaluation focuses on the ability of WEDAGEN to generate web tables of different complexities and sizes and the overheads incurred. In our experiments, the overhead of web table generation is measured by the *elapsed time* required to generate a web table.

As shown in Figure 7, four web schemas with different connectivity topologies have been used for the web tables to be generated. The topologies include a linear graph (`Schema1`), a tree (`Schema2`), a disconnected graph (`Schema3`) and a connected graph with multiple source nodes (`Schema4`). The four corresponding schema files are given in Figures 8, 9, 10 and 11.

To evaluate the overheads of web table generation, web tables of three sizes, i.e., small, medium and large, are generated for each of the above four schemas. In other words, there are altogether 12 web tables to be generated. The table sizes are determined by three sets of default parameters shown in Tables 1, 2 and 3. Note that a different parameter value for *NumSourceNodeInstances* is specified for each combination of web schema and web table size. This is necessary in order for WEDAGEN to generate sufficient node and link instances for constructing the required number of web tuples. Given a set of parameter values, number of tuples is determined by maximum number of sink node instances. Hence, the parameter values have been determined in a way to generate sufficient number of sink node instances to satisfy the number of tuples requirement.

To avoid the amount of user efforts involved in the experiments, each of the above 12 web tables is generated directly from the default parameter values, i.e., the specific parameter generation, instance generation, and tuple extraction modules are invoked automatically without user intervention.

Combining the four different web schemas and the three different web table sizes, we have conducted experiments that require WEDAGEN to generate the 12 web tables and measured the elapsed time for each web table generation. The experiments have been conducted on a lightly

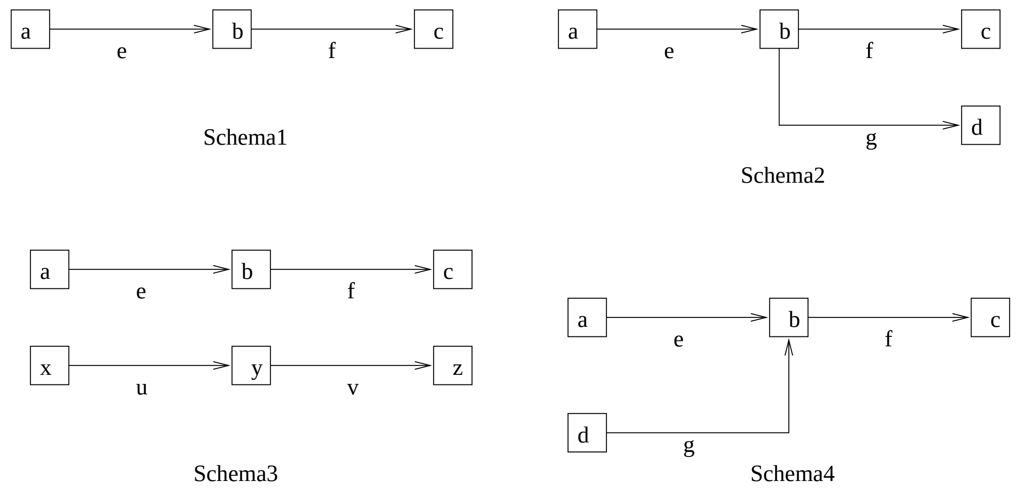

Figure 7: Topologies of Experimental Web Schemas.

```
[NODES]
a
   URL CONTAINS ‘‘departments’’
b
   title CONTAINS ‘‘electronics’’
c
   title CONTAINS ‘‘conference’’

[LINKS]
e
   label EQUALS ‘‘research’’
f
   label CONTAINS ‘‘publications’’

[CONNECTIVITIES]
a⟨e⟩b
b⟨f⟩c
```

Figure 8: The Schema File for `Schema1`.

```
[NODES]
a
   URL CONTAINS ``departments''
b
   title CONTAINS ``electronics''
c
   title CONTAINS ``conference''
d
   URL CONTAINS ``development''

[LINKS]
e
   label EQUALS ``research''
f
   label CONTAINS ``publications''
g
   label EQUALS ``laboratory''

[CONNECTIVITIES]
a⟨e⟩b
b⟨f⟩c
b⟨g⟩d
```

Figure 9: The Schema File for `Schema2`.

Parameter	**Value**
$NumTuples$	600
$NumSourceNodeInstances$	
- `Schema1`	24
- `Schema2`	1
- `Schema3`	1
- `Schema4`	24
$FanOut$	5
$NumWordsPerNodeInstance$	200
$NumWordsPerLinkLabel$	6
$NumWordsPerHostname$	5
$NumWordsPerTitle$	6
$LocalGlobalLink$	0
$NodeSelectivity$	60

Table 1: Parameter Values for a Small-Size Web Table

```
a
   URL CONTAINS ‘‘departments’’
b
   title CONTAINS ‘‘electronics’’
c
   title CONTAINS ‘‘conference’’
x
   URL CONTAINS ‘‘school’’
y
   title CONTAINS ‘‘group’’
z
   title CONTAINS ‘‘teachers’’

[LINKS]
e
   label EQUALS ‘‘research’’
f
   label CONTAINS ‘‘publications’’
u
   label EQUALS ‘‘laboratory’’
v
   label EQUALS ‘‘science’’

[CONNECTIVITIES]
a⟨e⟩b
b⟨f⟩c
x⟨u⟩y
y⟨v⟩z
```

Figure 10: The Schema File for Schema3.

```
[NODES]
a
   URL CONTAINS ``departments''
b
   title CONTAINS ``electronics''
c
   title CONTAINS ``conference''
d
   URL CONTAINS ``development''

[LINKS]
e
   label EQUALS ``research''
f
   label CONTAINS ``publications''
g
   label EQUALS ``laboratory''

[CONNECTIVITIES]
a⟨e⟩b
b⟨f⟩c
d⟨g⟩b
```

Figure 11: The Schema File for `Schema4`.

Parameter	**Value**
$NumTuples$	2000
$NumSourceNodeInstances$	
- `Schema1`	20
- `Schema2`	1
- `Schema3`	1
- `Schema4`	20
$FanOut$	10
$NumWordsPerNodeInstance$	100
$NumWordsPerLinkLabel$	10
$NumWordsPerHostname$	10
$NumWordsPerTitle$	10
$LocalGlobalLink$	0
$NodeSelectivity$	90

Table 2: Parameter Values for a Medium-Size Web Table

Parameter	**Value**
$NumTuples$	5000
$NumSourceNodeInstances$	
- `Schema1`	8
- `Schema2`	4
- `Schema3`	1
- `Schema4`	8
$FanOut$	25
$NumWordsPerNodeInstance$	200
$NumWordsPerLinkLabel$	10
$NumWordsPerHostname$	10
$NumWordsPerTitle$	10
$LocalGlobalLink$	0
$NodeSelectivity$	90

Table 3: Parameter Values for a Large-Size Web Table

loaded SUN workstation configured with 128MB RAM and 2×4.2GB hard disk space. The results of our experiments are shown in Figure 12.

Based on the results shown in Figure 12, the following observations can be made:

- As expected, the elapsed time of generating a web table increases with the size of the table. However, the rates of growth are different for web tables with different schemas. This indicates that the complexity of schema also affects the time taken to synthesize web tables. For example, the elapsed time of generating a web table with a tree schema, e.g. *Schema2*, is higher than compared to that with a linear schema, i.e. *Schema1*.

- There is a very slight increase in the elapsed time of *Schema3*. This is mainly due to fewer node and link instances required for generating a web table as compared to any other schema which requires more node and link instances for generating a web table of the same size. Moreover, *Schema3* is relatively simple involving fewer inter-connectivities between the node variables. Hence, the overhead of tuple generation is low.

- The elapsed time for *Schema2* is more than that for *Schema4*. This trend is explained as follows. *Schema4* can be obtained from *Schema2* by performing a reverse-link operation on link g. According to Theorem 1 in Section 5, the maximum number of sink node instances will increase after reversing the link. As the total number of tuples generated is directed related to the maximum number of sink node instances, to generate same number of tuples for both schemas, a smaller value of *NumSourceNodeInstances* should be assigned to *Schema4*. This therefore leads to fewer total number of node instances to be generated. Hence, to generate the same number of tuples for *Schema2* and *Schema4*, a smaller total number of node instances is required for *Schema4* as compared to *Schema2*. The larger number of node instances to be generated explains the higher elapsed time required for *Schema2*.

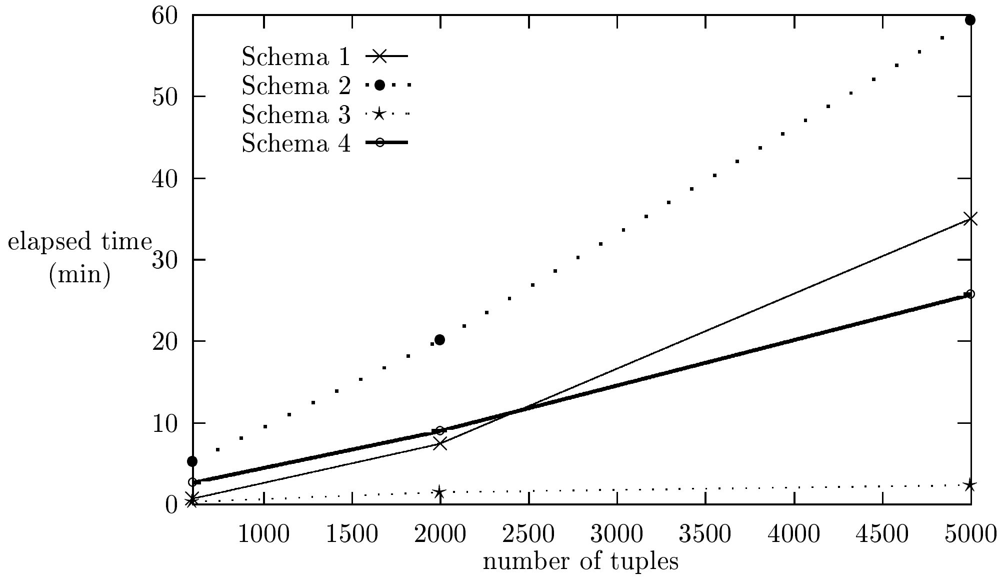

Figure 12: Elapsed Times for the Generation of 12 Web Tables.

8 Conclusions and Future Work

WEDAGEN is designed to generate synthetic web tables and web pages for comprehensive performance evaluation of web database systems. In this research, we have determined a set of parameters that allows web data of different complexities and sizes to be synthesized. The design and implementation of WEDAGEN have been described. A preliminary empirical evaluation of WEDAGEN indicates that the synthetic web database generator is able to scale up well with increasing web schema complexity and web table size.

With the availability of WEDAGEN, the time and efforts required to systematically evaluate the performance of web database systems can be reduced dramatically. One can quickly construct a specially calibrated web database for performance evaluation or testing purposes instead of having to download actual web pages from the WWW.

At present, WEDAGEN is being used in the WHOWEDA project to evaluate the performance of low level data access operations on the web tables. More work also remains to be done for WEDAGEN to make it more versatile. In particular, WEDAGEN can be extended to handle more varied web schema that may include unbound node and link variables, and more expressive predicates. The other major future works include:

- More parameters for web database generation can be designed for the future versions of WEDAGEN to improve upon the flexibility of web table generation. There are some more parameters that can be incorporated into WEDAGEN to achieve greater control over the generation process.

- WEDAGEN can be developed into a full-fledged benchmark toolkit for web database systems (not necessary WHOWEDA). To achieve this, a standard set of web queries and a benchmark web database have to be determined. In this aspect, WEDAGEN can be further explored to generate a benchmark web database. We plan to look into the design of both the benchmark web database and the benchmark web query set. With the complete benchmark toolkit, the performance of different web database systems can be compared and contrasted.

References

[1] T. Anderson, A.J. Berre, M. Mallison, and B. Schneider H.H. Porter. The hypermodel benchmark. In *International Conference on Extending Database Technology (EDBT)*, pages 317–331, Venice, Italy, March 1990.

[2] S. Bhowmick, W.-K. Ng, and E.-P. Lim. Join processing in web databases. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria, August 1998.

[3] S. Bhowmick, W.-K. Ng, E.-P. Lim, F. Qin, and X. Ye. A data warehousing system for web information. In *East Meets West: The First Asia Digital Library Workshop*, Hong Kong, August 1998.

[4] S.-S. Bhowmick, W.-K. Ng, and E.-P. Lim. Join processing in web databases. In *International Conference on Database and Expert Systems Applications*, Vienna, Austria, August 1998.

[5] S.S. Bhowmick, W.-K. Ng, and E.-P. Lim. Information coupling in web databases. In *International Conference on Conceptual Modeling (ER'98)*, Singapore, November 1998.

[6] H. Cameron and H. Traces. *Collection and Container Classes in C++*. Benjamin/Cumming Publishing Co., 1994.

[7] M.J. Carey, D.J. DeWitt, and J.F. Naughton. The 007 benchmark. In *ACM SIGMOD Conference*, Washington, D.C., November 1993.

[8] R.G.G. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, 17(1):1–31, 1992.

[9] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 Storage Benchmark. In *ACM SIGMOD Conference*, Washington, D.C., November 1993.