Formulating Disjunctive Coupling Queries in a Web Warehouse

Sourav S Bhowmick[a], Ang Kho Kiong[a] and Sanjay Madria[b]

[a]*School of Computer Engineering, Nanyang Technological University, Singapore 639798*

[b]*Department of Computer Science, University of Missouri-Rolla, Rolla 65409*

Abstract

We describe how to formulate a *coupling query* to glean relevant Web data in the context of our web warehousing system called WHOWEDA (*W*are*h*ouse *O*f *We*b *Da*ta). Coupling query may be used for querying both HTML and XML documents. One of the important feature of our query mechanism is the ability to express conjunctive as well as disjunctive query conditions compactly. We describe how to formulate a coupling query in text form as well as pictorially using the *coupling text* and the *coupling graph* respectively. We explore the limitations of coupling graph with respect to the coupling text. We found out that *AND*, *OR* and *AND/OR-coupling graphs* are less expressive than their textual counterparts. To address this shortcoming we introduce the notion of *hybrid graph* which is a special type of *p*-connected coupling graph. Finally, we discuss the implementation of a GUI-based system called VISCOUS (*VIS*ual *CO*upling Q*U*ery *S*ystem) for formulating such queries.

Key words: Web join, web warehouse, coupling query, coupling graph, query formulation, viscous

1 Introduction

The Web has invaded our lives. The exponential growth of the Web in the last few years had a significant impact on the traditional techniques used for data management during the last few decades. This has compelled the database community to reuse traditional techniques wherever possible to manage Web data. Unfortunately, due to the very nature of Web data, it is not always

Email addresses: {`assourav`}`@ntu.edu.sg`, `madrias@umr.edu` (Sanjay Madria).

System	Query HTML	Query XML	Querying hyper-links	Sub-page Querying	Partial knowledge	Querying tags, tag attributes	Control query exec.	Preserve result structure
Coupling Query	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
W3QS	Yes	No	Limited	Limited	Limited	No	No	No
WebSQL	Yes	No	Limited	No	No	No	No	No
WebLog	Yes	No	Limited	No	No	No	No	No
NetQL	Yes	No	Limited	Limited	Limited	No	Yes	No
FLORID	Yes	No	Limited	Limited	Limited	No	No	No
ARANEUS	Yes	No	Limited	Limited	Limited	No	No	No
WebOQL	Yes	No	Limited	Yes	Yes	No	No	Yes
Lorel	No	Yes	Not HTML link	Yes	Yes	Yes	No	Yes
XML-QL	No	Yes	Not HTML link	Yes	Yes	Yes	No	Yes
YAT$_L$	No	Yes	Not HTML link	Yes	Yes	Yes	No	Yes

Table 1
Comparison of web query systems.

possible to reuse conventional techniques effectively. This has led the database community to rethink and reuse existing techniques in a new way to address the current challenges. For instance, the database research community has devoted considerable attention to extend database querying techniques to data on the Web.

In this paper, we describe some of the features of a query mechanism for populating a data warehouse specifically for Web data, i.e., *web warehouse*. Broadly, we are interested in the three components of a web query mechanism:

(1) Determining the components, syntax, semantics and expressive power of the query language;
(2) Formulation techniques of a web query; and
(3) Evaluation procedure of the query.

We have studied (1) in [10] and (3) in [6,22] in the context of the *global web coupling* operation. This paper addresses the component (2). We introduce the notion of a *coupling query* to express a web query and show how it is formulated in the context of our web warehousing system, called WHOWEDA (*W*are*h*ouse *O*f *We*b *Da*ta) [5,22]. Note that the scope of this work is limited to textual documents. It does not include querying of images, video or other multimedia objects in the Web. Also this query mechanism cannot express constraints on forms that invoke CGI scripts. A shorter version of this paper appeared in [7].

1.1 Motivation

Currently, majority of Web data are in HTML format. However, in the near future more and more XML documents will coexist with HTML documents. Consequently, querying the Web implies querying large collection if interlinked HTML and XML documents. As a result a large body of research has been motivated by the attempt to extend database manipulation techniques to data on the Web [12,13,17,18] and several web query mechanisms such as W3QS [19], WebSQL [24], WebLog [20], RAW [16], NetQL [21], WebOQL [3], ULIXES in ARANEUS system [4], XML-QL [13], Lorel [18] and YAT_L [12] have been proposed so far. However, none of these query mechanisms completely address all the important features of a web query system. Table 1 gives a summary of the features of different web query systems. This led us to the design of coupling query which incorporates different important features of a web query mechanism into a single system. Note that in this paper our intention is not to perform an exhaustive comparison between the pros and cons of different web query systems with respect to the coupling query. We only highlight those features (discussed below) which are pivotal issues in this paper. The reader may refer to [10] for a detailed discussion on the comparison between existing web query techniques and coupling query.

Formulating web queries Intuitively, a web query represents a graph-like structure with constraints on some of its vertices and edges which are matched against the Web. Conventionally, graphs can be represented in text form as well as pictorially. Consequently, a web query may take both textual and pictorial forms. Textual formulation of web query enables us to express any complex web query accurately. As a result, most of the contemporary web query mechanism as well as query languages for semistructured data focused on a text-based query languages. However, text-based queries has some disadvantages. To express such queries, a user must be completely familiar with the syntax of the query language, and must be able to express his/her needs accurately in a syntactically correct form. Otherwise, a text-based query may be error-prone and may contain superfluous query conditions. Also due to the nature of Web data, specifying such query in text form requires considerable effort. For instance, query languages such as XML-QL, Lorel, YAT_L, WebOQL and FLORID, although are powerful languages, but is definitely not easy to formulate in textual form. Although, it is possible to apply syntactic sugar on these languages, but issues involved with such effort are not discussed in [13,18,12,3,23]. W3QS [19] allows us to use query templates to minimize the complexity associated with the formulation of web queries. Note that research on visual querying has been done in traditional database research [15,26]. To a greater or lesser extent, all these research focused on devising novel visual querying schemes to replace data retrieval aspects of SQL language. Specifically, forms have been popular building blocks for visual querying mechanism as well. For instance, Embley [15] proposed the NFQL as a communication language between humans and database systems. It uses forms in a strictly nonprocedural manner to represent query. As shall be seen in Section 4, we allow a user to formulate a coupling query both in text form and pictorially.

Disjunctive constraints on hyperlinked documents Hyperlinks are perhaps most important for relating parts of data that are not near each other in terms of prose flow. In the Web environment, the authors inclination to create many small pages, rather than single monolithic documents makes this even more important. Authors are motivated to create small pages to keep retrieval latencies low [25]. However, the structure of hyperlinked documents is irregular. Hence, it is important for a web query language to express disjunctive constraints on the hyperlinked structure compactly.

Informally, such disjunctive query may be decomposed into a set of conjunctive queries which are in disjunction to one another. A set of documents satisfies such disjunctive query if they satisfy any one of the conjunctive query conditions. Consequently, one may argue the justification of disjunctive queries because such queries can be represented by conjunctive query sets which are relatively easier to formulate. However, we believe that it is necessary for a web query mechanism to express disjunctive conditions for the following reasons:

- First, disjunctive queries allow us to overcome the limitations of irregular structure of inter-linked Web documents and pose meaningful queries over it.
- Second, sometimes query evaluation is relatively less expensive if a query is formulated and evaluated using disjunctive constraints rather than repeated evaluation of each query in the equivalent conjunctive query set.
- Third, expressing all possible set of conjunctive query accurately for a disjunctive condition incurs significant cognitive overhead which may result in erroneous query.
- Finally, a disjunctive query can be expressed compactly using regular expression. Expressing all possible set of conjunctive queries can be quite cumbersome and frustrating.

Although the importance of imposing disjunctive constraints on inter-document structure is undeniable, most of the web query systems support limited [12,13,17–19,24,20,16,21,3,4] form of disjunctive constraints, if any, on the inter-document structure of the Web. NetQL, WebSQL, W3QS, WebLog and FLORID do not address the issue of such constraints on interlinked documents extensively. A limited form of disjunctive condition which involves the variability of the depth of traversal of a query can be expressed by these languages. Additionally, these systems do not support querying of XML data. Query systems for semistructured data and XML query languages such as XML-QL, Lorel, YAT$_L$ also support limited form of disjunctive constraints on the inter-linked structure of Web documents. Query languages for semistructured data such as Lorel [1], UnQL [11] were not specifically developed for the Web, and do not distinguish between graph edges that represent the connection between a document and one of its parts and edges that represent a hyperlink from one Web document to another. On the other hand, XML query languages such as XML-QL, Lorel, YAT$_L$ support disjunctive conditions. However, whether these languages can express disjunctive conditions on the hyperlink structure of Web documents are not evident in [1,12,13,18]. Our coupling query allows us to express disjunctive constraints extensively using regular expressions in *connectivities* and *predicates*.

1.2 Paper Organization

The rest of the paper is organized as follows: In Section 2, we formally define coupling query. In Section 3, we introduce two types of coupling queries, i.e., *canonical* and *non-canonical*. Next, in Section 4 we discuss how to formulate canonical and non-canonical queries in text form and pictorially. These two forms of queries are called *coupling text* and *coupling graph* respectively. In Section 5, we discuss the implementation of a GUI-based system called VISCOUS (*VIS*ual *CO*upling q*U*ery *S*ystem) for formulating such queries. Finally, we conclude by summarizing this paper.

2 Coupling Query

We now formally introduce the notion of a *coupling query*. We begin by briefly describing the underlying data model of our web warehousing system. Then, we describe a model of information on the Web. Next, we illustrate with examples the components of a coupling query for expressing various query conditions. In Section 2.4, we formally define coupling query. In Section 3, we shall discuss two flavours of coupling query; *canonical* and *non-canonical*.

2.1 Data Model of WHOWEDA

Since our goal is to populate a web warehouse, we use as a starting point the WHOWEDA system. In a data warehouse designed for Web information it is imperative to represent and store relevant hyperlink Web documents effectively for further querying and manipulation. The *Ware*H*ouse* O*bject* M*odel*

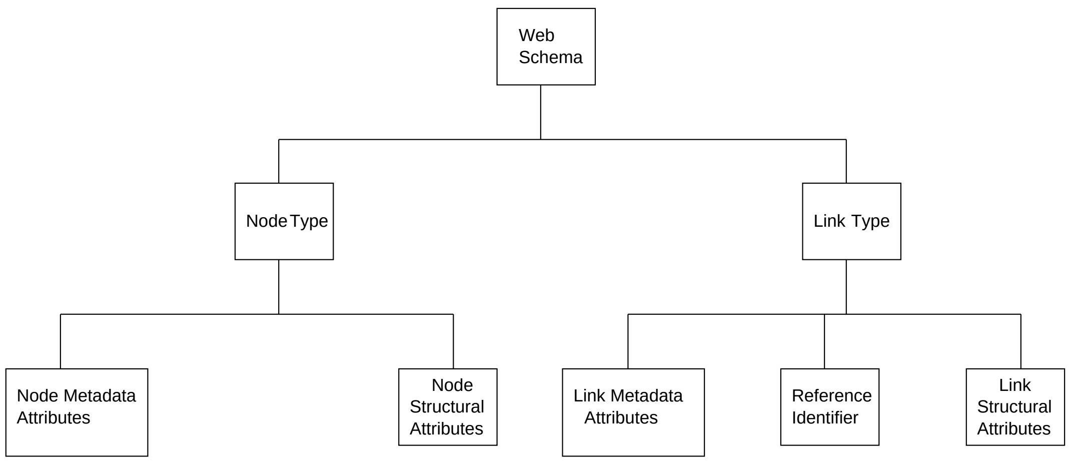


Fig. 1. Class Hierarchy.

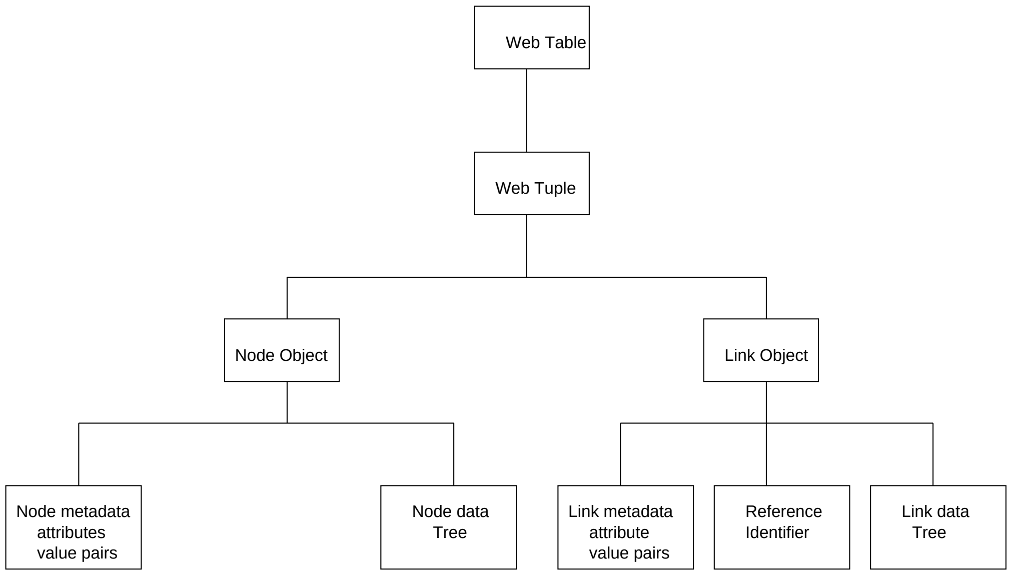


Fig. 2. Instance Hierarchy.

(WHOM) [5] serves as the basic data model for our web warehousing system. WHOM, like any other data model, consists of two components: a set of *web objects* and a set of *web operators*[5]. WHOM defines the logical structure of a set of objects in the web warehouse and the way these objects are accessed and manipulated.

Informally, our web warehouse can be conceived of as a collection of *web tables*. A set of *web tuples* is materialized in a web table. A web tuple is a directed graph consisting of sets of *node* and *link objects* (hereafter, referred to as *nodes* and *links* respectively for brevity).

In WHOWEDA, *nodes* and *links* are instances of *node type* and *link type* respectively. A *node type* consists of a *name*, a set of *node metadata attributes* and a set of *node structural attributes*. *Node metadata attributes* are used to capture the metadata information associated with Web documents (excluding hyperlinks) such as `URL`, `date` of last modification and `size`. Note that `URL` can be further decomposed into the following attributes: `server`, `protocol`, `path`, `filename` and `port`. On the other hand, the *node structural attributes* are used to represent the content and hierarchical structure of Web pages. Intuitively, a *node* represents the metadata associated with a Web document and the content and structure of the document (excluding hyperlinks in the document). Specifically, it consists of two components: a set of *node metadata trees* to represent values of different metadata associated with the document and a *node data tree* (directed labeled tree) to represent the content and structure of the document. A node metadata tree is an instance of a node metadata attribute and a node data tree is a set of *node structural objects* satisfying certain *dependency constraints*. The node structural objects are instances of node structural attributes. The notion of dependency constraints play an important role in determining the hierarchical relationships among node structural objects in a HTML or XML document. Similarly, a *link type* consist of a *name*, a set of *link metadata attributes*, a set of *link structural attributes* and a *reference identifier*. Thus, a *link* consists of a set of link meta-attribute/value pairs (such as `target URL`, `source URL` and `link_type`) represented as *link metadata trees*, a *link data tree* (instance of link structural attributes) and an unique reference identifier. Link data tree is a directed labeled tree to represent the structure and content of a HTML or XML link [1]. The reference identifier is used to associate the *location* of links in a particular Web document or node. Informally, one can think of a location as a portion of a document or a position in it. Observe that although a hyperlink is embedded in a Web document, we logically separate hyperlinks from Web documents while modeling HTML and XML data in WHOM. Figures 1 and 2 provide a pictorial representation of the hierarchical structure of the web objects in WHOM. Figure 3 represent the relationship between the web objects and its instances.

To facilitate manipulation of Web data stored in web tables, we have defined a set of web algebraic operators (i.e., *global web coupling*, *web join*, *web select* etc.) with web semantics. These web operators enables us to build new web tables by extracting relevant data from the Web and generate new web tables from the existing ones. Table 2 summarizes comparison of WHOM with respect to other web data models. The reader may refer to [5,9] for complete discussion on how Web data is represented and manipulated in the warehouse.

[1] We only consider simple and extended XML links.

System	Model HTML	Model XML	Data Model	Internal structure modeling	Metadata modeling	Content modeling	Hyper-link	Order	tag attri--bute	Mixed tag
WHOM	Yes	Yes	Labeled tree	Yes	Yes	Yes	Yes	Yes	Yes	Yes
W3QS	Yes	No	Labeled multigraph	Yes	Yes	Yes	Yes	No	No	No
WebSQL	Yes	No	Relational	No	Yes	Yes	Yes	No	No	No
WebLog	Yes	No	Relational	No	Yes	Yes	Yes	No	No	No
FLORID	Yes	No	F-Logic	Yes	Yes	Yes	Yes	No	No	No
ARANEUS	Yes	No	Page scheme	Yes	Yes	Yes	Yes	No	No	No
WebOQL	Yes	No	Hypertrees	Yes	No	Yes	Yes	Yes	No	No
STRUDEL	Yes	No	Labeled graph	Yes	No	Yes	Yes	Yes	No	No
Lore	No	Yes	Labeled graph	Yes	No	Yes	XML-link	Yes	Yes	No
XML-QL	No	Yes	Labeled graph	Yes	No	Yes	XML-link	Yes	Yes	Yes

Table 2
Comparison of web data models.

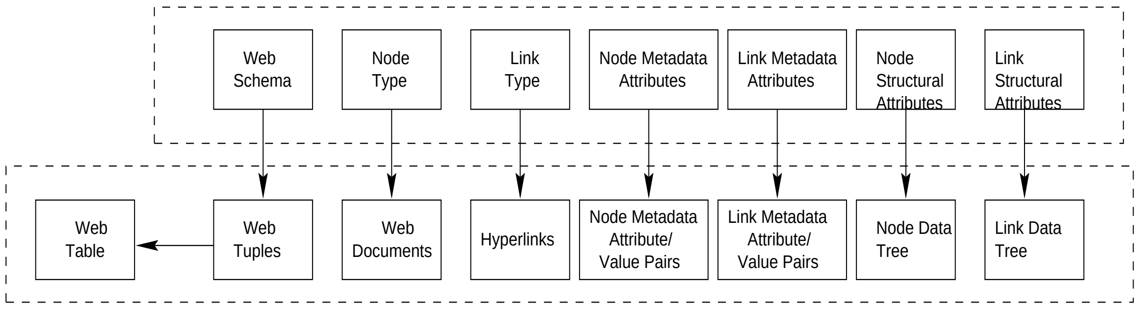


Fig. 3. Relationship between logical and instance level.

2.2 The Information Space

The WWW involves a large number of information spaces ranging from simple files to complex service providers that are distributed over the Internet. In order to formally deal with information, we need to define a conceptually unified information space against which users can formulate queries. We view the WWW as a directed graph. The entire graph topology is unknown but can be partly deduced by navigating the Web. The vertices and edges of the graph are defined by every possible WWW navigation activity.

We assume that the WWW is deterministic [19]. By this, we mean that the WWW structure, content and programs (i.e., CGI scripts) are static and that programs are deterministic and time independent. This is clearly a simplification of the WWW. However, it allows us to assume that the WWW does not change during the execution of a query. The following definition captures the hypertext structure of WWW accessible information.

Definition 1 *A* **WWW Graph** $G(WWW) = (V, E)$ *is a pair where* V *and* E *are sets of node and link objects on the Web and* $E \subseteq V \times V$*. Each edge* $e \in E$ *is a hyperlink from a node object* $v \in V$ *to* $u \in V$ *such that there is a link object from a node object* v *to*

- *any node object corresponding to a file accessible by clicking on a valid hyperlink in* v*,*
- *any node object corresponding to the data returned by filling a form in* v *and*
- *the default error HTML message (Error 404) obtained by clicking on an invalid hyperlink in* v*.* ∎

Observe that the previous definition captures a simplified model of the actual WWW. This is done for the sake of simplicity. Our model can be easily extended to include data accessible through protocols other than `http` and more complex HTML constructs such as `frames`. Although, we do not discuss the querying and processing of HTML forms in this paper, our model can be extended to handle them.

2.3 Components of a Coupling Query

A coupling query consists of the following five components:

A set of *node* and *link type identifiers* X_n and X_ℓ respectively. Each nominal identifier in X_n or X_ℓ represents a set of documents or hyperlinks (possibly empty) retrieved from the Web (also called *node* and *link objects*). Each identifier in X_n or X_ℓ may be either *bound* or *free*. The set of node or link objects represented by a *bound* type identifier share some *common properties* in terms of their metadata, content or structure. Some of these properties are expressed explicitly in the query using a set of *predicates*. To elaborate further, let G be a coupling query and P be the set of predicates in G. Let d_1, d_2, $\ldots$, d_n be a set of documents represented by a node type identifier $d \in X_n$ in G. Then d is bound type identifier if there exist a set of predicates $P_d \subseteq P$ defined over d. Each predicate in P_d specifies metadata, content or structural characteristics shared by the documents d_1, d_2, $\ldots$, d_n. On the other hand, a *free* type identifier do not have any predicate defined over it in the coupling query. That is, there are no conditions in terms of metadata, content or structure imposed by the user on the nodes or links represented by the free type identifier. In Whoweda, we denote such free node and link type identifiers by using the special symbols '`#`' and '`-`' respectively. Note that the instances of free type identifiers represent arbitrary nodes or links.

A set of *predicates* P on the node and link type identifiers to express the conditions defined by a user that must be satisfied by the relevant documents and hyperlinks of the corresponding identifiers. The following is a form of a predicate p on x:

$$p(x) \equiv \texttt{predicate\_qualifier::}x\{\texttt{[attribute\_path\_exp]}\}$$
$$\texttt{predicate\_operator "}V\texttt{"}$$

where x, the argument of p, is a node or link type identifier depending on the application of constraints on Web documents or hyperlinks respectively. The component `predicate_qualifier` determines the *scope* of the predicate. It can have any one of the following values: "`METADATA`", "`CONTENT`" and "`STRUCTURE`". It indicates whether the predicate is to be used to impose constraints on the metadata, textual content or structure of instances of x. The component `attribute_path_exp` is essentially a sequence of tags which may include wild cards and regular expression operators. It is used to specify constraints on specific position(s) of a Web document and hyperlink. It may also be used to impose structural constraints on Web data. `Predicate_operator` represents operators such as `EQUALS`, `ATTR_ENCL`, `NON-ATTR_ENCL`, `ATTR_CONT`, `NON-ATTR_CONT` and `CONT` to test for string regular expression matching. The operators containing the strings `ATTR` and `NON-ATTR` are used to distinguish between the attribute/value pairs associated with tags of HTML or XML elements and the textual content between tagged elements when desired. The strings `CONT` and `ENCL` in these operators are used to further distinguish between partial and complete data segments in an element or in the attribute set associated with an element. Operators such as `SATISFIES` and `EXISTS_IN` are used to impose conditions on the structure. The operator `SATISFIES` checks if the instances of a node or link type identifier satisfies a particular element structure. The operator `EXISTS_IN` is specifically used on link objects and checks if instances of a link type identifier exist in the specified portion of source documents. V is called the *value* of the predicate and is a regular expression over the ASCII character set when the `predicate_qualifier` is either `METADATA` or `CONTENT`. When the qualifier is `STRUCTURE`, V may be an attribute path expression or a collection of tag element names. The curly brackets are used to indicate 0 or 1 occurrence of the components.

A collection of *connectivities* C (in conjunctive or disjunctive form)[2] to express the hyperlink structure the relevant documents must satisfy with respect to the user's query. Informally, a *connectivity* is a predicate on the inter-document relationship of one or two classes of Web documents. To define a

[2] Traditionally, in the field of graph theory, the term connectivity of a connected graph refers to the minimum number of vertices whose removal disconnects the graph or reduces the graph to a single vertex. However, in this paper, we do not use the notion of connectivities in this context.

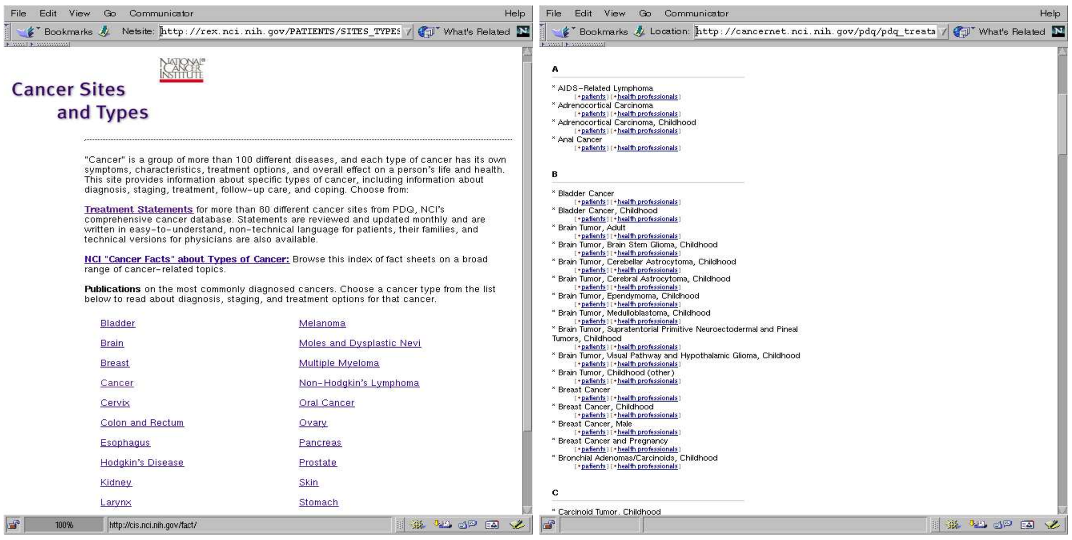
connectivity element, one first categorizes the set of documents and hyperlinks into different types by using a set of predicates. Then connectivities are defined by using the type identifiers of these documents and hyperlinks. A connectivity k is an expression of the form: $k \equiv \texttt{s}\langle\rho\rangle\texttt{t}$ where `s` is the *source node type identifier* (*source identifier* in short), `t` is the *target node type identifier* (*target identifier* in short) and ρ is called a *link path expression* which is essentially a sequence of link type identifiers which may include regular expressions, e.g., e, efg, $ef\{1,3\}$. The angle brackets around ρ are used for delimitation purposes only. Note that the connectivity $\texttt{s}\langle\rho\rangle\texttt{t}$ specifies how the instances of `s` are connected to the instances of `t`. The interlinked structure between an instance of `s` and `t` identifier is specified by the link path expression. Throughout this paper, we denote the source and target identifiers of a connectivity k as $lnode(k)$ and $rnode(k)$ respectively. The set of link type identifiers in ρ is denoted as $link(k)$.

A connectivity element is categorized into two types—*simple* and *complex*. A *simple* connectivity contains only *simple* link path expression. By simple link path expression we mean that there is no regular expressions defined over it. Hence, a simple connectivity contains only one link type identifier in the link path expression. For instance, $x\langle e\rangle y$ is a simple connectivity. On the other hand, in a complex connectivity, the link path expression may contain regular expressions. For instance, $x\langle efgh\rangle y$, $x\langle(ef?)|(g\text{-})\rangle y$, $x\langle e\text{-}\{1,5\}\rangle y$ are examples of complex connectivities.

A set of *conditions* Q on the coupling query to control the execution of the query for retrieving relevant data.

2.3.1 Our Approach

We now illustrate these components by formulating a coupling query. Consider the NCI Web site at `rex.nci.nih.gov`. Suppose a user wishes to retrieve information related to treatment of different types of cancer. This site provides information about specific types of cancer, including information about diagnosis, staging, treatment, follow-up care and coping. Specifically, links in the Web page at `rex.nci.nih.gov/PATIENTS/SITES_TYPES.html` provide links to information related to different types of cancer. The link "treatment statement" points to a page containing a list of links to cancer-related diseases (Figure 4(b)). Each of these links point to a page containing information on diagnosis, treatments and so on of a particular disease. There are also hyperlinks labeled "bladder", "brain", and so on in the Web page at `rex.nci.nih.gov/PATIENTS/SITES_TYPES.html` (Figure 4(a)) which directly connects to a page containing details of these diseases. Observe that some of the links such as "AIDS-related lymphoma", "Anal Cancer", "Endometrical Cancer" and so on in Figure 4(b) are not available in the Web page at


File Edit View Go Communicator Help
Bookmarks
Netsite: http://rex.nci.nih.gov/PATIENTS/SITES_TYPES
What's Related
100%
http://cis.nci.nih.gov/fact/
File Edit View Go Communicator Help
Bookmarks
Location: http://cancernet.nci.nih.gov/pdq/pdq_treatm
What's Related

NATIONAL CANCER INSTITUTE

Cancer Sites and Types

"Cancer" is a group of more than 100 different diseases, and each type of cancer has its own symptoms, characteristics, treatment options, and overall effect on a person's life and health. This site provides information about specific types of cancer, including information about diagnosis, staging, treatment, follow-up care, and coping. Choose from:

Treatment Statements for more than 80 different cancer sites from PDQ, NCI's comprehensive cancer database. Statements are reviewed and updated monthly and are written in easy-to-understand, non-technical language for patients, their families, and technical versions for physicians are also available.

NCI "Cancer Facts" about Types of Cancer: Browse this index of fact sheets on a broad range of cancer-related topics.

Publications on the most commonly diagnosed cancers. Choose a cancer type from the list below to read about diagnosis, staging, and treatment options for that cancer.

Bladder	Melanoma
Brain	Moles and Dysplastic Nevi
Breast	Multiple Myeloma
Cancer	Non-Hodgkin's Lymphoma
Cervix	Oral Cancer
Colon and Rectum	Ovary
Esophagus	Pancreas
Hodgkin's Disease	Prostate
Kidney	Skin
Larynx	Stomach

A

* AIDS-Related Lymphoma
  [+patients] [+health professionals]
* Adrenocortical Carcinoma
  [+patients] [+health professionals]
* Adrenocortical Carcinoma, Childhood
  [+patients] [+health professionals]
* Anal Cancer
  [+patients] [+health professionals]

B

* Bladder Cancer
  [+patients] [+health professionals]
* Bladder Cancer, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Adult
  [+patients] [+health professionals]
* Brain Tumor, Brain Stem Glioma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Cerebellar Astrocytoma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Cerebral Astrocytoma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Ependymoma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Medulloblastoma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Supratentorial Primitive Neuroectodermal and Pineal Tumors, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Visual Pathway and Hypothalamic Glioma, Childhood
  [+patients] [+health professionals]
* Brain Tumor, Childhood (other)
  [+patients] [+health professionals]
* Breast Cancer
  [+patients] [+health professionals]
* Breast Cancer, Childhood
  [+patients] [+health professionals]
* Breast Cancer, Male
  [+patients] [+health professionals]
* Breast Cancer and Pregnancy
  [+patients] [+health professionals]
* Bronchial Adenomas/Carcinoids, Childhood
  [+patients] [+health professionals]

C

* Carcinoid Tumor, Childhood

(a) Treatment for cancer. (b) List of cancer related disease.

Fig. 4. Web pages of NCI Web site.

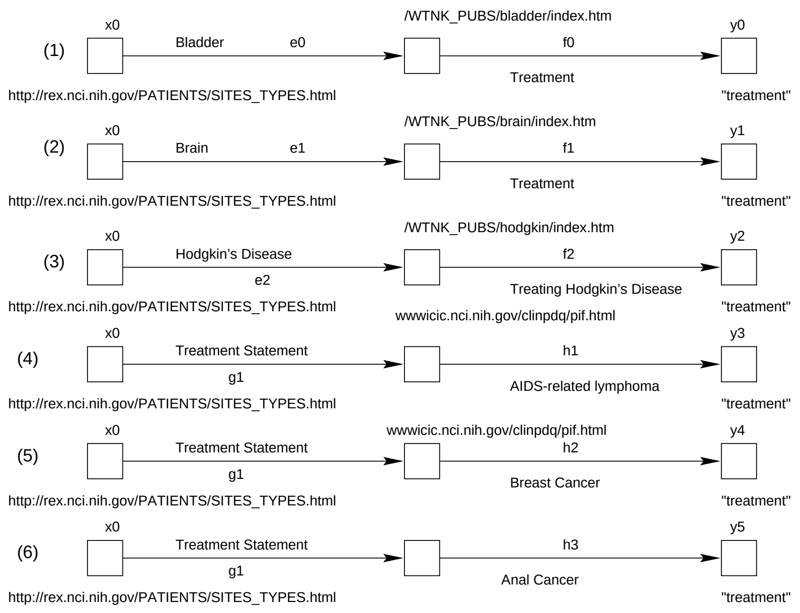
`http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html`. Similarly, links related to "Non-Hodgkin's Lymphoma", "Hodgkin's Disease", "Stomach" in the Web page in Figure 4(a) are not listed in the Web page in Figure 4(b). However, links related to "Larynx", "Melanoma" and so on are available in both Web pages. Hence, in order to retrieve a complete list of treatment details of various types of cancer we need to exploit the link "treatment statement" and links related to different cancers in the Web page in Figure 4(a). In order to express this query, we need:

- A starting point for the search (Web page at `rex.nci.nih.gov/PATIENTS/SITES_TYPES.html`)
- to scan the pages accessible from the starting page by following links having the specific characteristics as described above.

Therefore,

(1) We search for a path in the Web hypertext structure, beginning at the Web page at `rex.nci.nih.gov/PATIENTS/SITES_TYPES.html` and ending at a page containing the keyword "treatment" by following only hypertext links that satisfies the above conditions. Such hypertext path can be expressed in the coupling query by the *connectivity* $x\langle(ef)|(gh)\rangle y$. Here x, y are *node type identifiers* and e, f, g and h are *link type identifiers*. Observe that the expression $(ef)|(gh)$ enables us to express disjunctive conditions in a coupling query, i.e., either follow the links of types e and f or follow the links of types g and h.

(2) Instances of x is the first vertex of the path and corresponds to the page at



(1)
x0
Bladder
e0
/WTNK_PUBS/bladder/index.htm
f0
Treatment
y0
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"
(2)
x0
Brain
e1
/WTNK_PUBS/brain/index.htm
f1
Treatment
y1
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"
(3)
x0
Hodgkin's Disease
e2
/WTNK_PUBS/hodgkin/index.htm
f2
Treating Hodgkin's Disease
y2
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"
(4)
x0
Treatment Statement
g1
wwwicic.nci.nih.gov/clinpdq/pif.html
h1
AIDS-related lymphoma
y3
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"
(5)
x0
Treatment Statement
g1
wwwicic.nci.nih.gov/clinpdq/pif.html
h2
Breast Cancer
y4
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"
(6)
x0
Treatment Statement
g1
h3
Anal Cancer
y5
http://rex.nci.nih.gov/PATIENTS/SITES_TYPES.html
"treatment"

Fig. 5. Results of the query in Example 2

`rex.nci.nih.gov/PATIENTS/SITES_TYPES.html`. Coupling query allows the mapping of specific pages to a node type identifier. This is written in the form of a *predicate*:

$p_{1_1}(x) \equiv$ `METADATA::x[url] EQUALS` "*rex[.]nci[.]nih[.]gov/PATIENTS/SITES_TYPES[.]html*"

(3) The link type identifiers e, f, g and h must satisfy the above conditions. These conditions are expressed as following predicates:

$p_{1_3}(f) \equiv$ `STRUCTURE::f[A] EXISTS_IN` "*body.p*"
$p_{1_4}(e) \equiv$ `CONTENT::e[A] NON-ATTR_ENCL` "*Treatment Statements*"
$p_{1_5}(h) \equiv$ `CONTENT::h[A] ATTR_CONT` "{*(href, MATCH(:BEGIN_STR: + treat.* + :END_STR:)*}"
$p_{1_6}(g) \equiv$ `STRUCTURE::g SATISFIES` "*A*"
$p_{1_7}(g) \equiv$ `STRUCTURE::g[A] EXISTS_IN` "*table(.%)+.p*"

The first predicate specifies that the instances of f exist in a paragraph contained in the `body` element of the source documents. The next predicate specifies that the anchor text of instances of e is *Treatment Statements*. The predicate $p_{1_5}(h)$ says that the instances of h must contain the element `A` having an attribute labeled `href`. The value of `href` must match the regular expression *treat.**. That is, the target URL of the hyperlinks must contain the string "treat". The last two predicates indicate that g is a link type identifier and specifies that instances of g exist in a paragraph contained in the `table` elements of the source documents. Note that similar to node type identifier, specific hyperlinks are mapped to a link type identifier. Also, observe that the predicates allow us to impose constraints on specific portions of Web documents or hyperlinks, on attributes associated with HTML or XML elements and on the hierarchical structure of Web documents based on partial knowledge of the structure of the documents.

(4) The last vertex y must contain the keyword "treatment" anywhere in the document and is expressed by the following predicate:

```
p_12(y) ≡ CONTENT::y[html(.%)+] NON-ATTR_CONT
          ":BEGIN_WORD: + treatment + :END_WORD:"
```

(5) In order not to overload the `rex.nci.nih.gov` HTTP server, we limit the time taken for the search. The search stops after 20 minutes and returns the result retrieved so far. Also we make sure that all the documents retrieved by the search belongs to the Web site of NCI. This is done by defining the following *coupling query predicates*:

```
q_1(G_1) ≡ COUPLING_QUERY::G_1.time EQUALS "20 min"
q_2(G_1) ≡ COUPLING_QUERY::G_1.host EQUALS "rex.nci.nih.gov"
```

Note that coupling query predicates enables us to control the execution of a query.

The query is then expressed as:

Example 2 Let the coupling query be $G_1 = \langle X_{n_{1q}}, X_{\ell_{1q}}, C_{1q}, P_{1q}, Q_{1q} \rangle$

$$X_{n_{1q}} = \{x, y\}$$
$$X_{\ell_{1q}} = \{e, f, g, h\}$$
$$C_{1q} \equiv x\langle(ef)|(gh)\rangle y$$

$P_{1q} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ where

```
p_11(x) ≡ METADATA::x[url] EQUALS
          "rex[.]nci[.]nih[.]gov/PATIENTS/SITES_TYPES[.]html"
p_12(y) ≡ CONTENT::y[html(.%)+] NON-ATTR_CONT
          ":BEGIN_WORD: + treatment + :END_WORD:"
p_13(f) ≡ STRUCTURE::f[A] EXISTS_IN "body.p"
p_14(e) ≡ CONTENT::e[A] NON-ATTR_ENCL "Treatment Statements"
p_15(h) ≡ CONTENT::h[A] ATTR_CONT
          "{(href, MATCH(:BEGIN_WORD: + treat.* + :END_WORD:)}"
p_16(g) ≡ STRUCTURE::g SATISFIES "A"
p_17(g) ≡ STRUCTURE::g[A] EXISTS_IN "table(.%)+.p"
```

and $Q_{1q} = \{q_1, q_2\}$ where

```
q_1(G_1) ≡ COUPLING_QUERY::G_2.time EQUALS "20 min"
q_2(G_1) ≡ COUPLING_QUERY::G_1.host EQUALS "rex.nci.nih.gov"
```

The set of query results is shown in Figure 5. Observe that the results are directed connected graph and preserves the hyperlinked structure of the relevant documents that satisfies the query conditions. These graphs are called *web tuples*. Intuitively, a web tuple t is a subgraph of $G(WWW)$ and the set of documents and hyperlinks in t satisfies the connectivities and predicates defined in a query G. ∎

2.4 Definition of Coupling Query

We now formally define a coupling query. Note that although a coupling query consists of five components as described in the preceding section, any arbitrary sets of node and link type identifiers, connectivities and predicates do not represent a valid coupling query. The following conditions must be satisfied by the components of a coupling query:

Conditions on Node and Link Type Identifiers The conditions outlined below must be satisfied by the X_n and X_ℓ components of the coupling query.

- The set of node type identifiers X_n is always non-empty and must contain at least one bound node type identifier. That is, $X_n \neq \emptyset$.
- The identifiers used to represent node objects in the coupling query must be nominally dissimilar to those used to represent link objects. That is, the components X_n and X_ℓ must not overlap; i.e., $X_n \cap X_\ell = \emptyset$.

Conditions on Connectivities We now identify the constraints imposed on the collection of connectivities in a coupling query.

- The set of node and link type identifiers in the collection of connectivities must match with the set of node and link type identifiers specified in the components of X_n and X_ℓ.
- The next condition specifies the only case when a coupling query may not contain any connectivities. This is possible only when the set of node objects are represented by a single node type identifier. In this case the predicate set is non-empty as the node type identifier cannot be a free type identifier.
- If C_1 and C_2 represents two *conjunctive connectivity collections* in the coupling query and $C_1 \vee C_2$ then C_1 and C_2 must not contain the same collection of connectivities.

Condition on Predicates Finally, the predicate set in a coupling query must satisfy the following condition: The argument of each predicate in the predicate set must be a node or link type identifiers in X_n or X_ℓ.

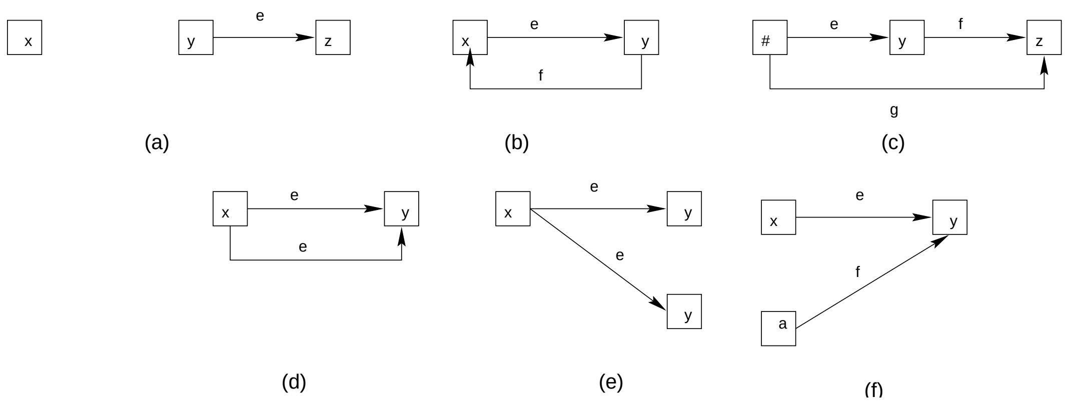


Fig. 6. Graphical representation of invalid coupling queries.

Topological Conditions on the Coupling Query A connectivity can be visualized as a directed connected acyclic graph. As a coupling query contains a collection of connectivities, it can also be visualized as a directed graph, where a vertex and an edge of the graph are labeled by a node and link type identifier respectively and a set of predicates, if any, on these identifiers. Furthermore, to simplify formulation of coupling query and for its efficient computation we pose certain constraints on the graphical form of a coupling query as outlined below:

- The graphical view of a coupling query must be a directed connected acyclic graph. This indicates that not only each connectivity in a coupling query is a connected DAG, but also the union of all the connectivities in a coupling query must also be a connected DAG. We do not allow coupling query to be disconnected as such queries may not be *computable* [2]. For instance, Figure 6(a) is an example of disconnected graph and hence does not represent a valid coupling query. Also we do not allow cycles in a coupling query in order to simplify the evaluation of the query. Figure 6(b) represents an invalid coupling query containing a cycle.

  It may seem that forbidding cycles in a coupling query would significantly hinder the expressiveness of a web query. This is indeed true as cycles are very common in the Web. Web sites may specifically be designed to distribute information across pages, and provide extensive links to lead the user back and forth across these pages. In fact, proper usage of cycles in a Web site is considered to be one of the important feature for good Web site design. Hence, to address this problem we follow a two-level approach. At the first level, we use an acyclic coupling query to gather relevant documents from the Web and materialize them in our web warehouse. In the second level, we use the *web select* operation to impose cyclic constraints on the materialized query results [5].
- The graphical representation of a coupling query must have a single source vertex. That is, there must be only one vertex with zero in-degree. We

disregard queries with multiple source vertices to simplify formulation and evaluation of coupling queries. Figure 6(f) is an example of an invalid query with multiple source vertices.
- The source vertex of the query must always be bound. That is, the node type identifier in the query representing the source vertex must always be associated with a set of non-trivial predicates defined over it. This is to ensure computability of the query. Figure 6(c) is an example of an invalid query with free source vertex.
- The labels of two edges in the graph are identical only if the start and end vertices pair of each edge is not identical to one another. Hence, the query in Figure 6(d) is invalid.
- The labels of two vertices in the graph can be identical only if the labels of the incoming edges and their start vertices are not identical. The query in Figure 6(e) is invalid because of the violation of this rule.

Based on the above features, a coupling query can be formally defined as follows:

Definition 3 [**Coupling Query**] *A* **coupling query** *is a 5-tuple* $G = \langle X_n, X_\ell, C, P, Q\rangle$ *where* X_n *is a set of node type identifiers,* X_ℓ *is a set of link type identifiers,* C *is a collection (possibly empty) of connectivities in conjunctive or disjunctive form defined over* X_n *and* X_ℓ*,* P *is a set of predicates defined over* X_n *and* X_ℓ *and* Q *is a set (possibly empty) of coupling query predicates such that the following conditions are true:*

- $X_n \neq \emptyset$*,* $P \neq \emptyset$*,* $X_n \cap X_\ell = \emptyset$*;*
- *If* $|X_n| = 1$ *then* $X_\ell = \emptyset$*,* $C = \emptyset$ *and* $P \neq \emptyset$*;*
- *Let* X_{nc} *and* $X_{\ell c}$ *be the set of node and link type identifiers in* C *respectively. Then* $X_{nc} = X_n$ *and* $X_{\ell c} = X_\ell$*;*
- *There must not exist conjunctive connectivity collection* $C_a \equiv k_{a1} \wedge k_{a2} \wedge \cdots \wedge k_{an}$ *and* $C_b \equiv k_{b1} \wedge k_{b2} \wedge \cdots \wedge k_{bn}$ *such that* $C_a \vee C_b$ *and* $k_{ax} = k_{bx}$ $\forall$ $0 < x \leqslant n$*;*
- *Let* $p(x) \in P$*. Then* $x \in (X_n \cup X_\ell)$*;*
- *Let* $G(C)$ *be the graphical representation of* C*. Then* $G(C)$ *must be a directed connected acyclic graph with single source vertex. Further, let* x *be the identifier of the source vertex in* $G(C)$*. Then, there must exist a non-trivial predicate* $p(x) \in P$*.* ∎

3 Canonical and Non-canonical Coupling Queries

A coupling query is categorized into two types: *canonical* and *non-canonical*. This categorization is based on the type of connectivities in the query and their relationship with one another. We elaborate on this further.

We say that a coupling query is *canonical* if it contains a set (possibly empty) of simple connectivities in Disjunctive Normal Form (DNF). For example, if $G = \langle X_n, X_\ell, C, P, Q \rangle$, $C \equiv C_1 \vee C_2$ where $C_1 \equiv k_1 \wedge k_2$, $C_2 \equiv k_3$ $k_1 \equiv x\langle e \rangle y$, $k_2 \equiv y\langle f \rangle z$ and $k_3 \equiv x\langle e \rangle z$ then G is a canonical coupling query. C_1 and C_2 are called *conjunctive connectivity sets*. Based on the above definition of canonical coupling query, we classify canonical queries into the following five types. Let $G_c = \langle X_n, X_\ell, C, P, Q \rangle$ be a canonical coupling query. Then,

- **Type 1:** G_c does not contain any connectivities. That is, $|X_n| = 1$, $X_\ell = \emptyset$ and $C = \emptyset$. Note that this is the simplest form of coupling query.
- **Type 2:** G_c contains a single simple connectivity. That is, $|X_n| = 2$, $|X_\ell| = 1$, $C \equiv k$ where k is a simple connectivity.
- **Type 3:** G_c contains more than one simple connectivities and these connectivities are in conjunction. That is, $C \equiv k_1 \wedge k_2 \wedge \cdots \wedge k_r$ where k_i is a simple connectivity for all $1 < i \leqslant r$.
- **Type 4:** G_c contains more than one simple connectivities and these connectivities are in disjunction. That is, $C \equiv k_1 \vee k_2 \vee \cdots \vee k_r$ where k_i is a simple for all $1 < i \leqslant r$.
- **Type 5:** G_c contains more than one simple connectivities and these connectivities are in DNF. That is, $C \equiv C_1 \vee C_2 \vee \cdots \vee C_r$ where C_i is a conjunctive connectivity set for all $1 < i \leqslant r$.

A *non-canonical coupling query*, on the other hand, may contain simple or complex connectivities and these connectivities may not be in DNF. For instance, if $C \equiv k_1 \wedge k_2$, $k_1 \equiv x\langle e|f \rangle y$ and $k_2 \equiv y\langle g \rangle z$ then G is a non-canonical coupling query. This is because k_1 is a complex connectivity. We classify non-canonical queries into the following four types. Let $G_{nc} = \langle X_n, X_\ell, C, P, Q \rangle$ be a non-canonical coupling query. Then,

- **Type 1:** G_{nc} contains a single complex connectivity. That is, $|X_n| = 2$, $X_\ell \neq \emptyset$, $C \equiv k$ where k is a complex connectivity.
- **Type 2:** G_{nc} contains more than one connectivity and at least one of them is complex. Further, these connectivities are in conjunction. That is, $C \equiv k_1 \wedge k_2 \wedge \cdots \wedge k_r$ where k_i is complex for $1 < i \leqslant r$.
- **Type 3:** G_{nc} contains more than one connectivity and at least one of them is complex. Further, these connectivities are in disjunction. That is, $C \equiv k_1 \vee k_2 \vee \cdots \vee k_r$ where k_i is complex for $1 < i \leqslant r$.
- **Type 4:** G_{nc} contains more than one simple or complex connectivities and these connectivities are in conjunction and in disjunction to one another.

Note that a user may specify any form of coupling query to harness relevant documents from the Web. For brevity, different types of canonical and non-canonical coupling queries are denoted as G_c^t or G_{nc}^t respectively where $0 < t \leqslant 5$ and indicates G_c or G_{nc} is of Type t.

We now illustrate canonical and non-canonical coupling queries with examples.

Example 4 Consider the query in Example 2. Recall that a user wishes to retrieve information related to treatment of different types of cancer from the NCI Web site. This query is non-canonical as the connectivity is complex in nature. ∎

Next, we provide an example of canonical form of coupling queries.

Example 5 Consider the non-canonical coupling query in Example 2. This can be expressed as a canonical query by reducing the complex connectivities into sets of simple connectivities in DNF. The formal representation of the query is as follows: Let the canonical coupling query be $G_2 = \langle X_{n_{2q}}, X_{\ell_{2q}}, C_{2q}, P_{2q}, Q_{2q} \rangle$ where $X_{n_{2q}} = \{x, y, \#_1, \#_2\}$, $X_{\ell_{2q}} = \{e, f, g, h\}$, $C_{2q} \equiv C_a \vee C_b$, $C_a \equiv x\langle e\rangle\#_1 \wedge \#_1\langle f\rangle y$, $C_b \equiv x\langle g\rangle\#_2 \wedge \#_2\langle h\rangle y$, $P_{2q} = P_{1q}$ and $Q_{2q} = Q_{1q}$. Notice that this is an example of canonical Type 5-coupling query as all the connectivities are simple and are in DNF. ∎

3.1 Valid Canonical Coupling Query

A *valid* canonical coupling query is necessary for generating *web schemas* [8] of a set of web tuples retrieved from the Web. Moreover, this form is also necessary for *global web coupling* operation [22]. Informally, a canonical coupling query is valid if each conjunctive connectivity set in the query represents a directed connected acyclic graph with single source vertex. Hence, a canonical query must satisfy the following conditions;

Directed Connected Graph with Single Source Vertex Each conjunctive connectivity set must represent a directed connected graph with single source vertex. We illustrate this condition with an example. Consider C_1 to be a conjunctive connectivity set in a canonical coupling query. Let $k_1 \equiv a\langle e\rangle b$ be a connectivity in C_1. Then there must exist another connectivity of the form $a\langle f\rangle z$, $x\langle f\rangle b$ or $x\langle f\rangle a$ in C_1 if the number of simple connectivities in C_1 is more than one. Furthermore, if a represents the source vertex then $x\langle f\rangle b$ or $x\langle f\rangle a$ cannot exist in C_1. Also the node type identifier represented by the source vertex must be bound.

Acyclic Condition on Connectivities Each conjunctive connectivity set must represent an acyclic graph. Hence, if $x\langle e\rangle y$ is a connectivity in C_i then there must not exist another connectivity $y\langle f\rangle x$ in C_i. Observe that these two connectivities create a cycle. Secondly, if $k_1 \equiv x\langle e\rangle y$ be a connectivity such that there exist another connectivity $k_5 \equiv z\langle f\rangle x$ then this may result in a cycle if k_1 is connected to k_5 through a collection of connectivities (say k_2, k_3 and k_4).

Acyclic Conditions on Predicates Next, we discuss conditions on predicates in a coupling query for ensuring the query to be acyclic in nature. We first illustrate the conditions with an example. Let x and y be two node type identifiers such that $|P_x| > 1$ or $|P_y| > 1$, $p_1(x) \in P_x$, $p_1(y) \in P_y$ and

$p_1(x) \equiv$ `METADATA::x[url] EQUALS "http://www[.]druginfonet[.]com"`
$p_2(y) \equiv$ `METADATA::y[url] EQUALS "http://www[.]druginfonet[.]com"`

Note that in this case the query may contain cyclic component if any one of the following conditions is not satisfied:

- If x and y belongs to a conjunctive connectivity set C_i then x and y must represent two adjacent identifiers. That is, if x and y are adjacent node type identifiers then instances of x and y represent identical documents connected by an interior link. Consequently, there must exist a simple connectivity $x\langle\ell\rangle y$ in C_i to express adjacency of these node type identifiers.
- Otherwise, both x and y does not exist in C_i. That is if $x \in C_i$ and $y \in C_j$ then $i \neq j$, i.e., C_i and C_j represent two conjunctive connectivity sets and $C_i \vee C_j$. Hence, C_i and C_j do not generate cyclic graphs.

3.2 Transformation of Non-Canonical Coupling Query

Recall that a user can express a coupling query in any form. In case, he/she pose a non-canonical query, we transform the query to a canonical form and *prune* it (if necessary) before it is evaluated by the global web coupling operation. For instance, the query in the Example 2 is transformed to its canonical form as shown in Example 5 before they are evaluated. Note that we do not discuss the procedure and issues involved with the transformation and *pruning* of a non-canonical coupling query to a canonical form in this paper as these issues will only increase the length of the paper without contributing substantially to the discussion of the mechanism to populate a web warehouse. The following proof is also omitted for space reasons. The reader may refer to [6] for detailed discussion.

Theorem 1 *Every non-canonical coupling query can be transformed to a valid canonical coupling query.* ∎

4 Coupling Query Formulation

As a coupling query is defined by a user, the structure and content of a coupling query depends on the following factors: First, the information a user wishes to retrieve from the Web. Second, the user's level of knowledge of the content and structure of the Web site(s) containing the relevant information. By default, a coupling query is formulated in text form. It may also be formulated graphically. The textual representation of the query is called *coupling text* and the pictorial representation of a coupling query is called *coupling graph*.

In coupling text, the user specifies the five components X_n, X_ℓ, C, P and Q in textual form. Coupling text is a flexible query formulation mechanism and can be used to specify any meaningful query. In the remaining portion of this paper we shall use coupling text and coupling query interchangeably. Examples 2 and 5 are examples of coupling text.

Next, we describe the second mechanism for formulating coupling queries, i.e., *coupling graph*. We begin by defining a coupling graph. Then we discuss different types of coupling graph a user may wish to draw. Next, we discuss the limitations associated with coupling graphs in expressing different forms of query. Finally, in Section 4.4 we introduce the notion of *hybrid graph* in order to resolve these limitations. Also note that, unless explicitly stated otherwise, a canonical coupling query indicates a valid canonical query.

4.1 Definition of Coupling Graph

Informally, a coupling graph is a directed connected acyclic graph. This mechanism enables a user to specify a coupling query by drawing a graph. The label of vertices of the graph are node type identifiers and predicates, if any, defined over these identifiers. The label of the edges of the graph are link type identifiers and predicates on these link type identifiers (if any). The predicates are specified by clicking on the vertices and edges. The edges between the vertices specifies the connectivity constraints. The set of coupling query predicates is specified by clicking on the entire coupling graph.

A coupling graph is used to express queries containing simple connectivities only. We justify the reasons behind this. Recall that complex connectivity is a compact mechanism when expressed in text form for expressing a set of simple connectivities which are in conjunction or in disjunction to one another. Thus, coupling text containing complex connectivities enable a user to specify a query tersely without having the overhead of expressing all possible form of simple connectivities. However, this advantage of complex connectivities in coupling text cannot be realized when formulating the query using coupling graph. Essentially, a complex connectivity condense a set of node and link type identifiers into a single expression. Such capability cannot be realized when drawing a graph. To express the connectivities one has to draw all the edges and vertices. For instance, to express the connectivity $x\langle(ef)|(gh)\rangle y$ using coupling graph, the user has to draw all the vertices and edges as shown in Figure 9(b). This is equivalent to specifying all the simple connectivities which $x\langle(ef)|(gh)\rangle y$ represents. Hence, there is no additional advantage in allowing a user to draw a complex connectivity. For this reason we do not allow users to specify queries containing complex connectivities using a coupling graph. We believe coupling text is the best mechanism to express queries containing complex connectivities. Formally, the definition of a coupling graph is as follows:

Definition 6 [Coupling Graph] *A coupling graph $G_{cg} = (V_q, E_q)$ for a query $G = \langle X_n, X_\ell, C, P, Q\rangle$ is a connected acyclic digraph with single source vertex where*

- *C is a set of simple connectivities.*
- *V_q is a finite set of vertices. A vertex v_q is labeled by a node type identifier $id(v_q) \in X_n$ and a set (possibly empty) of predicates $P_n \subseteq P$ on the node type identifier. Furthermore, $V_q = V(k_1) \cup V(k_2) \cup \cdots \cup V(k_n)$ where $k_1, k_2, \ldots, k_n$ are connectivities in C, $G(k_i) = (V(k_i), E(k_i))\ \forall\ 0 < i \leqslant n$*
- *E_q is a finite set of directed edges such that $E_q = E(k_1) \cup E(k_2) \cup \cdots \cup E(k_n)$. An edge e_q is labeled by the link type identifier $id(e_q) \in X_\ell$ and a set (possibly empty) of predicates $P_\ell \subseteq P$.*
- *$g : E_q \rightarrow V_q \times V_q$ is a function such that $g(e_q) = (v_{q_1}, e_q, v_{q_2})$ if and only if there exist a simple connectivity $\mathrm{id}(v_{q_1})\ \langle\ \mathrm{id}(e_q)\ \rangle\ \mathrm{id}(v_{q_2})$ in C.* ∎

4.2 Types of Coupling Graph

We classify coupling graphs into three categories, i.e., *AND-coupling graph*, *OR-coupling graph* and *AND/OR-coupling graph*. We elaborate on these three types of coupling graph.

4.2.1 AND-Coupling Graph

In an *AND-coupling graph* all the edges are AND together. It is used to express pictorially a coupling query containing a set of simple connectivities in conjunction to one another (canonical queries of Types 2 and 3). Formally, let $G_{cg} = (V_q, E_q)$ be a coupling graph. Then G_{cg} is an AND-coupling graph if $e_{qi} \in E_q$ and $e_{qj} \in E_q$ and $e_{qi} \wedge e_{qj}\ \forall\ 0 < i, j \leqslant |E_q|$ and $i \neq j$. Note that the graphical representation of canonical form of coupling text (of Types

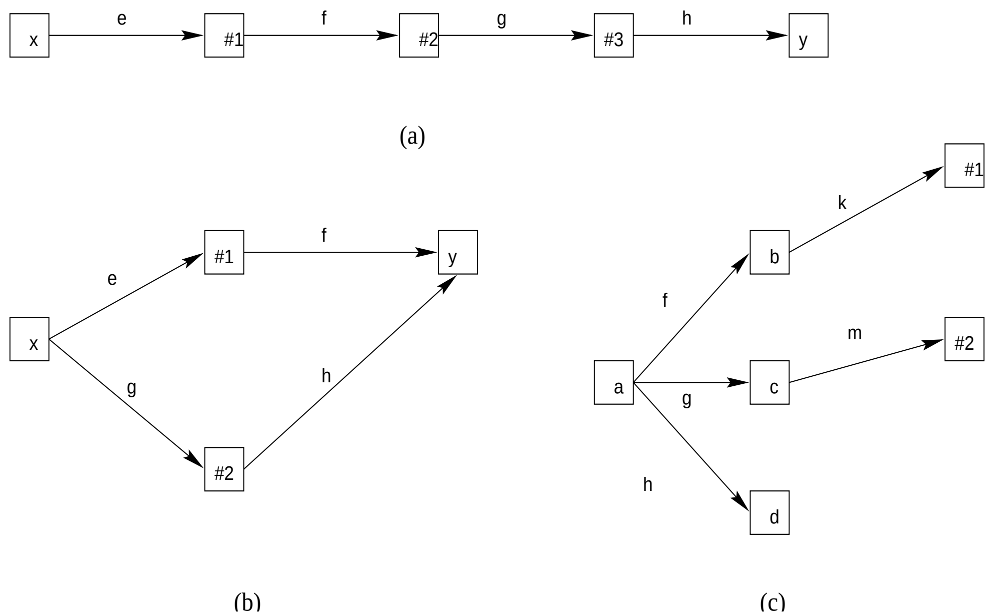


Fig. 7. AND-Coupling Graphs.

2 and 3) is identical to the corresponding AND-coupling graph. For example, Figures 7(a), 7(b) and 7(c) are examples of AND-coupling graphs expressing the sets of simple connectivities $(x\langle e\rangle\#_1 \wedge \#_1\langle f\rangle\#_2 \wedge \#_2\langle g\rangle\#_3 \wedge \#_3\langle h\rangle y)$, $(x\langle e\rangle\#_1 \wedge \#_1\langle f\rangle y \wedge x\langle g\rangle\#_2 \wedge \#_2\langle h\rangle y)$ and $(a\langle f\rangle b \wedge a\langle g\rangle c \wedge a\langle h\rangle d \wedge b\langle k\rangle\#_1 \wedge c\langle m\rangle\#_2)$ respectively.

4.2.2 OR-Coupling Graph

An *OR-coupling graph* is used to formulate pictorially coupling queries in which the connectivities are simple and are in disjunction to one another. In an *OR-coupling graph* all the edges are OR'd together. Note that OR-coupling graph cannot be linear as it requires at least two outgoing or incoming edges to be OR'd together. Furthermore, as we only allow coupling graphs with single source vertex, OR-coupling graphs must not have more than one vertex with no incoming edges. Consequently, OR-coupling graph pictorially represents queries containing a set of simple connectivities having identical source identifier. Moreover, these connectivities must generate a directed connected acyclic graph. As we disregard formulation of non-canonical coupling queries using coupling graphs, an OR-coupling graph is a pictorial representation of a canonical form of Type 4-coupling text. Formally, let $G_{cg} = (V_q, E_q)$ be a coupling graph. Then G_{cg} is an OR-coupling graph if $e_{qi} \in E_q$ and $e_{qj} \in E_q$ and $e_{qi} \vee e_{qj}$ $\forall$ $0 < i, j \leqslant |E_q|$ and $i \neq j$. Observe that the depth of an OR-graph is always equal to one. This is because each simple connectivity represents a path of length one. Hence, a set of simple connectivities in disjunction represents a graph having depth one. For

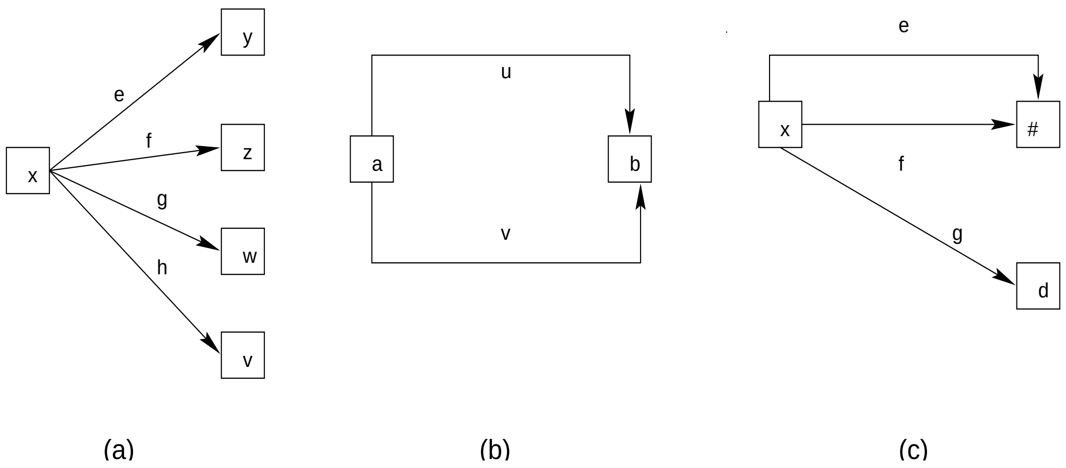


Fig. 8. OR-Coupling Graphs.

example, Figures 8(a), 8(b) and 8(c) are examples of OR-coupling graphs expressing connectivities $(x\langle e\rangle y \vee x\langle f\rangle z \vee x\langle g\rangle w \vee x\langle h\rangle v)$, $(a\langle u\rangle b \vee a\langle v\rangle b)$ and $(x\langle e\rangle\# \vee x\langle f\rangle\# \vee x\langle g\rangle d)$ respectively.

4.2.3 AND/OR-Coupling Graph

Informally, an AND/OR-coupling graph represents coupling queries in which the connectivities are in conjunction as well as in disjunction to one another. We first define the notion of *AND-edges* and *OR-edges* in order to elaborate on this type of coupling graph. In a coupling graph, an edge (v_1, e, v_2) is an *AND-edge* if the out-degree and in-degree of v_1 and v_2 respectively is equal to one. For example, in Figure 9(a) edges (y, f, a) and (z, h, b) are AND-edges. Otherwise, the edge is called an *OR-edge*. For instance, in Figure 9(a) the out-degree of vertex x is two. Hence, edges (x, e, y) and (x, g, z) are OR-edges. Now we define AND/OR-coupling graph. Let $G_{cg} = (V_q, E_q)$ be a coupling graph. Then, G_{cg} is an AND/OR graph if any one of the following conditions is true:

- If all edges are OR-edges then the depth of the graph must be greater than one. This is because if the depth is equal to one then the graph is an OR-coupling graph.
- G_{cg} must contain AND and OR-edges.

The significance of the above restriction regarding the definition of an AND/OR-coupling graph shall be best understood in Section 4.3.

Note that in an AND/OR-coupling graph all the outgoing or in-coming OR-edges to a vertex is OR'd together. Furthermore, connectivities in each level in the graph is AND together with the connectivities in the next level. For

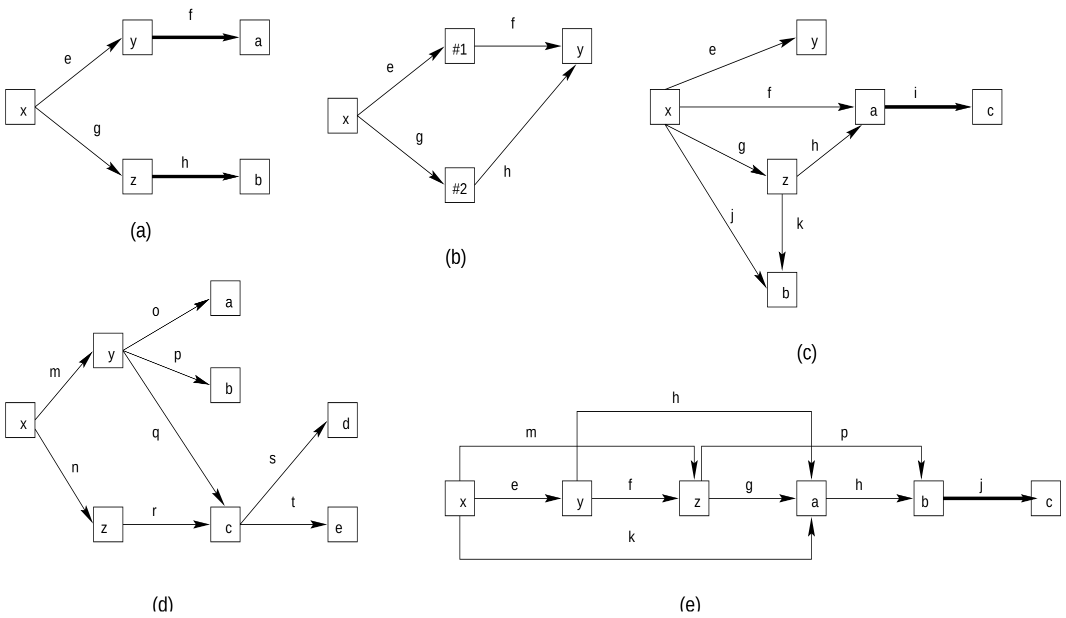


Fig. 9. AND/OR-Coupling Graphs.

example, Figure 9 shows a set of AND/OR-coupling graphs. In this figure, AND and OR-edges are shown by thick and thin arrows respectively. In Figure 9(a), the edges e and g are OR-edges and f and h are AND-edges. Hence, it expresses a coupling query with the following connectivities: $(x\langle e\rangle y \wedge y\langle f\rangle a)$ $\vee$ $(x\langle g\rangle z \wedge z\langle h\rangle b)$. Similarly, Figure 9(b) is the pictorial representation of the canonical coupling text in Example 5. Note that in this case all edges are OR-edges. Figure 9(c) expresses a query with the following connectivities: $x\langle e\rangle y \vee (x\langle f\rangle a \wedge a\langle i\rangle c) \vee (x\langle g\rangle z \wedge z\langle h\rangle a \wedge a\langle i\rangle c) \vee (x\langle g\rangle z \wedge z\langle k\rangle b) \vee x\langle j\rangle b$. Similarly, Figures 9(d) and 9(e) express the following connectivities: $(x\langle m\rangle y \wedge y\langle o\rangle a) \vee (x\langle m\rangle y \wedge y\langle p\rangle b) \vee (x\langle m\rangle y \wedge y\langle q\rangle c \wedge c\langle t\rangle e) \vee (x\langle m\rangle y \wedge y\langle q\rangle c \wedge c\langle s\rangle d)$ $\vee$ $(x\langle n\rangle z \wedge z\langle r\rangle c \wedge c\langle s\rangle d) \vee (x\langle n\rangle z \wedge z\langle r\rangle c \wedge c\langle t\rangle e)$ and $(x\langle e\rangle y \wedge y\langle f\rangle z \wedge x\langle g\rangle a \wedge a\langle h\rangle b \wedge b\langle j\rangle c) \vee (x\langle m\rangle z \wedge z\langle g\rangle a \wedge a\langle h\rangle b \wedge b\langle j\rangle c) \vee (x\langle k\rangle a \wedge a\langle h\rangle b \wedge b\langle j\rangle c) \vee$ $(x\langle e\rangle y \wedge y\langle f\rangle z \wedge z\langle p\rangle b \wedge b\langle j\rangle c) \vee (x\langle e\rangle y \wedge y\langle h\rangle a \wedge a\langle h\rangle b \wedge b\langle j\rangle c)$ respectively.

An AND/OR-coupling graph can be used to pictorially represent *some types* of canonical coupling text of Type 5, but not all. This is due to the inherent limitations of drawing an AND/OR-coupling graph to represent a unique coupling query. We discuss this issue in detail in the next subsection.

4.3 Limitations of Coupling Graphs

In the preceding sections we have described how to draw a canonical coupling query using coupling graph. In this section we explore the limitations of coupling graphs in expressing canonical coupling queries compared to its textual counterpart. In particular, we provide answer to the following question: Let G be a coupling query. Then, is it possible to express G by a coupling graph? That is, we explore the issue whether it is always possible to express any valid coupling query using a coupling graph.

4.3.1 Criteria

We first identify the criteria which is important in determining the limitations of coupling graph compared to its textual counterpart. Recall that predicates in a coupling query facilitates imposing constraints on metadata, content or structure of nodes and links. Furthermore, the ability to specify predicates in a coupling text and coupling graph is same. Whatever predicates that may be expressed in textual form in a coupling text, can also be expressed graphically in a coupling graph. Hence, the predicates do not play pivotal role in differentiating the expressiveness of these two types of query mechanism. For similar reasons, coupling query predicates do not influence the differences between the expressiveness of coupling text and coupling graph.

However, the connectivities in a coupling query plays a major role in the context of expressive power of coupling text and coupling graph. This is because coupling text allows us to express both simple and complex connectivities and enables us to impose explicitly how these connectivities are associated to one another. Consequently, it is possible to express the hyperlink structure precisely in a coupling text. On the other hand, the usage of connectivities in a coupling graph is restricted due to *certain limitations* of drawing a query. As a matter of fact, coupling graph can only express simple connectivities. Further, as we shall see a coupling graph does not provide the flexibility of expressing any combination of disjunctive relationship between a set of simple connectivities. This has a direct impact on the expressiveness of a coupling graph. In the following sections, we shall discuss the effect of connectivities on coupling graphs.

4.3.2 Limitations

In this section, we discuss the limitations of the three types of coupling graphs compared to their textual counterparts. We begin with AND-coupling graph.

AND-Coupling Graph Edges in an AND-coupling graph represents a set of simple connectivities AND together. This is equivalent to a set of simple connectivities in conjunction to one another in a canonical form of Types 2 and 3-coupling text. Hence, the expressiveness of AND-coupling graph is

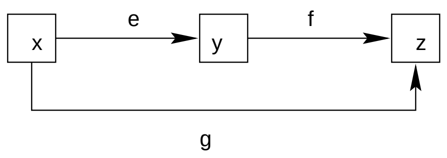


Fig. 10. Limitations of OR-coupling graph.

equivalent to the canonical form of Types 2 and 3-coupling text. Any query that can be expressed by canonical form of Types 2 or 3-coupling text can also be formulated using an AND-coupling graph.

OR-Coupling Graph An OR-coupling graph represent a set of simple connectivities in disjunction to one another. This is equivalent to the canonical form of Type 4-coupling text. Observe that each connectivity in the coupling query must represent a path from the source vertex to a leaf vertex in the OR-coupling graph. Hence, an OR-coupling graph cannot express those simple connectivities which represents a path other than those between the source vertex and leaf vertices in the graphical representation of the connectivities. We elaborate on this with an example. Consider the connectivities $C \equiv k_1 \vee k_2 \vee k_3$ where $k_1 \equiv x\langle e\rangle y$, $k_2 \equiv y\langle f\rangle z$ and $k_3 \equiv x\langle g\rangle z$. A query containing these connectivities can be expressed by a canonical form of Type 4-coupling text. The graphical representation of this query is shown in Figure 10. However, this query cannot be composed using an OR-coupling graph. This is because k_2 represents a path between an interior vertex and a leaf vertex in Figure 10. Typically, an OR-coupling graph is only capable of expressing simple connectivities with identical source identifiers. If all the source identifiers are not identical in the connectivities then there may exist a connectivity which does not represent a path between the source and leaf vertex. Consequently, we may conclude that the expressiveness of OR-coupling graph is not equivalent to that of the canonical form of Type 4-coupling text. Formally,

Condition 1 *Let $C \equiv k_1 \vee k_2 \vee \cdots \vee k_n$ be a set of simple connectivities in a valid canonical coupling query G_c^4. Then, G_c^4 cannot be expressed by an OR-coupling graph if* $\text{lnode}(k_i) \neq \text{lnode}(k_j)$ *for* $0 < (i,j) \leqslant n$ *and* $i \neq j$. ∎

AND/OR-Coupling Graph We now identify the limitations of AND/OR-coupling graphs and illustrate them with examples. Within this context we shall justify the reasons for the restrictive definition of an AND/OR-coupling graph as highlighted in Section 4. We begin with the first limitation.

Case 1 Recall that in an AND/OR-coupling graph each path from the source vertex to a leaf vertex represents a conjunctive connectivity set in the coupling

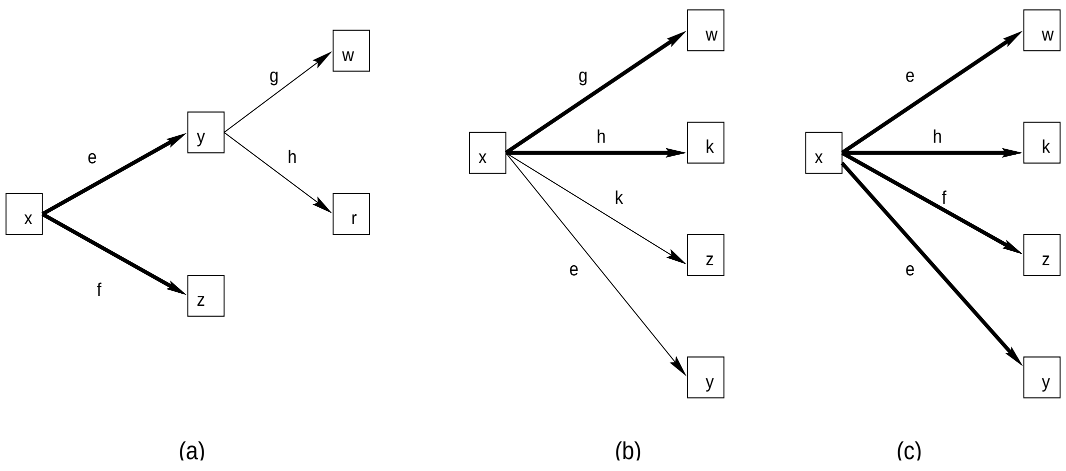


Fig. 11. Case 1.

query. In other words, if $C \equiv C_1 \vee C_2 \vee \cdots \vee C_n$ be a collection of connectivities in a coupling query G and if G_{cg} be the AND/OR-coupling graph of G then $C_1, C_2, \ldots, C_n$ represents such paths in G_{cg}. This indicates that an AND/OR-coupling graph fails to express a collection of connectivities that can be visualized as a non-linear structure, i.e., tree or a graph. For instance, an AND/OR-coupling graph cannot express the following connectivities: $C_3 \equiv C_{31} \vee C_{32}$ where $C_{31} \equiv x\langle e\rangle y \wedge x\langle f\rangle z \wedge y\langle g\rangle w$ and $C_{32} \equiv x\langle e\rangle y \wedge x\langle f\rangle z \wedge y\langle h\rangle r$. Similarly, it cannot express $C_4 \equiv C_{41} \vee C_{42}$ where $C_{41} \equiv x\langle g\rangle w \wedge x\langle h\rangle k \wedge x\langle f\rangle z$ and $C_{42} \equiv x\langle e\rangle y \wedge x\langle g\rangle w \wedge x\langle h\rangle k$. Observe that C_{31}, C_{32}, C_{41} and C_{42} can be visualized as trees.

One may argue that the above limitation arises because of the restrictive definition of AND/OR-coupling graph. That is, in an AND/OR-coupling graph if a vertex has more than one incoming or outgoing edges then these edges are OR'd together. We only allow an edge $e = (v_1, \ell, v_2)$ to be an AND-edge if the out-degree and in-degree of v_1 and v_2 respectively is equal to one. However, in order to accommodate the ability to express tree or graph structured connectivities using AND/OR-coupling graph it is imperative to allow AND-edges for vertices with more than one incoming or outgoing edges. It may seem that by relaxing the definition of an AND/OR-coupling graph it may be possible to resolve this limitation. For example, consider the graph in Figure 11(a). Let edges with identifiers e and f be AND-edges in lieu of OR-edges. Then C_3 in the above example can be expressed by this AND/OR-coupling graph. Similarly, the coupling graph in Figure 11(b) may be used to express the connectivities C_4.

Although it may seem that the resolution of this problem lies in the relaxation of the definition of AND and OR-edges, but so is not the case. Consider the connectivities $C_5 \equiv C_{51} \vee C_{52}$ where $C_{51} \equiv x\langle e\rangle w \wedge x\langle h\rangle k$ and

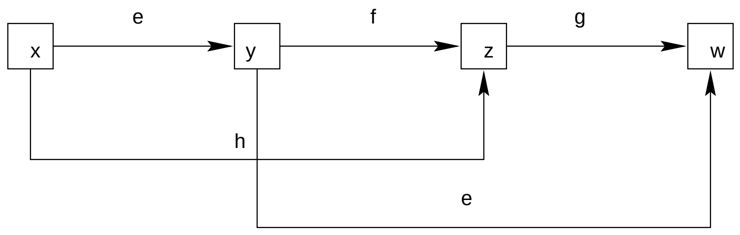


Fig. 12. Case 2.

$C_{52} \equiv x\langle f\rangle z \wedge x\langle e\rangle y$. The coupling graph of a query involving C_5 is shown in Figure 11(c). However, since all the edges are AND-edges, this graph actually express $(x\langle e\rangle w \wedge x\langle h\rangle k \wedge x\langle f\rangle z \wedge x\langle e\rangle y)$. This connectivity constraint cannot be expressed even by relaxing the definition of AND-edges. Hence, allowing such flexible definition of AND-edges may generate an AND/OR-coupling graph which may not represent the intended connectivities when transformed to its textual form. Due to this problem, we disallow AND-edges from vertices whose in-degree or out-degree is more than one. Formally,

Condition 2 *Let $G_c^5 = \langle X_n, X_\ell, C, P, Q\rangle$ be a valid canonical form of Type 5-coupling text where $C \equiv C_1 \vee C_2 \vee \cdots \vee C_r$. Let $G(C_i) = (V_i, E_i)$ be the graphical representation of C_i for $0 < i \leqslant r$. Then G_c^5 cannot be composed using an AND/OR-coupling graph if $G(C_i)$ is a tree or graph where $|V_i| > 2$.* ∎

Case 2 This shortcoming is similar to the one discussed in the context of OR-coupling graph. An AND/OR-coupling graph, similar to OR-coupling graph cannot express connectivity that represents a path in the graph other than those from the source vertex to leaf vertices. For example, consider the connectivities $C_6 \equiv C_{61} \vee C_{62} \vee C_{63}$ where $C_{61} \equiv x\langle e\rangle y \wedge y\langle f\rangle z \wedge z\langle g\rangle w$, $C_{62} \equiv x\langle h\rangle z \wedge z\langle g\rangle w$ and $C_{63} \equiv y\langle e\rangle w$. The graphical representation of these connectivities is shown in Figure 12. Observe that a query containing these connectivities cannot be expressed using an AND/OR-coupling graph. This is because C_{63} represents a path from an interior vertex to a leaf vertex in Figure 12. Formally,

Condition 3 *Let $G_c^5 = \langle X_n, X_\ell, C, P, Q\rangle$ be a valid canonical form of Type 5-coupling text where $C \equiv C_1 \vee C_2 \vee \cdots \vee C_r$. Let $C_i \equiv k_{i1} \wedge k_{i2} \wedge \cdots \wedge k_{iq}$ for all $0 < i \leqslant r$. Let $x = \mathrm{lnode}(k_{ij})$, $0 < j \leqslant q$ such that $x \neq \mathrm{rnode}(k_{is})$ $\forall$ $0 < s \leqslant q$ and $s \neq j$. Then G_c^5 cannot be expressed by an AND/OR-coupling graph if there exist C_t such that $y = \mathrm{lnode}(k_{tj})$, $y \neq lnode(k_{ts})$ and $y \neq x$, $t \neq i$.* ∎

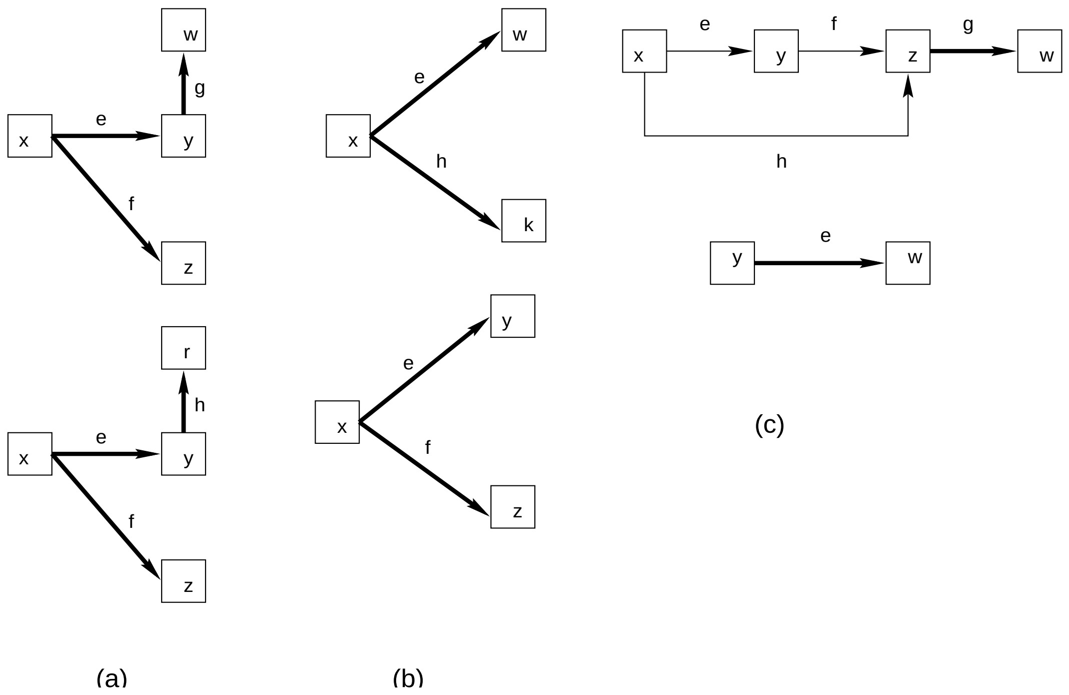


Fig. 13. Hybrid graphs.

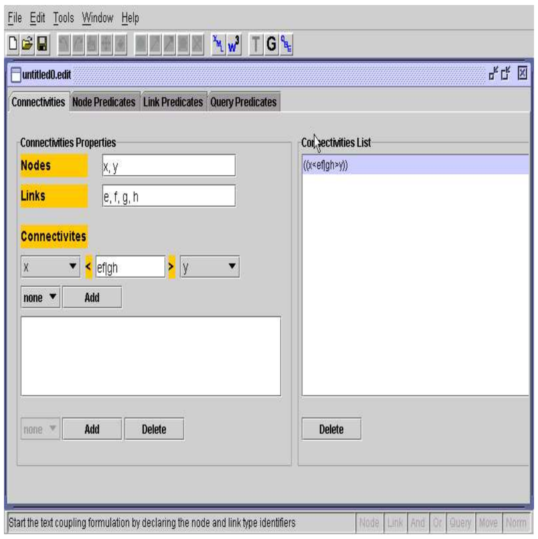
4.4 Hybrid Graph

We now introduce the notion of *hybrid graph* to resolve the limitations of OR and AND/OR-coupling graphs discussed in the preceding section. Informally, a *hybrid* graph is composed by drawing a p-connected coupling graph for $p > 1$ such that

- Each *connected component* [3] is an AND, OR or AND/OR-coupling graph. Each connected component in the hybrid graph is a set of vertices such that all vertices in the set are connected from the source vertex (reachable by some path).
- the connected components are in disjunction to one another and
- If $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ be two connected components then $V_i \cap V_j \neq \emptyset$.

For example, Figure 13(a) represents a hybrid graph with two connected components consisting of AND-coupling graphs. This hybrid graph represents the

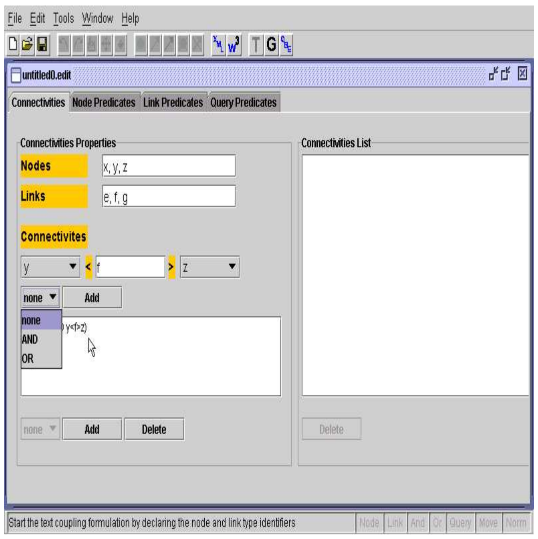
[3] Traditionally, given a graph $G = (V, E)$, where V is a set of vertices (of size n) and E is a set of edges (of size m), the connected components of G are the sets of vertices such that all vertices in each set are mutually connected (reachable by some path), and no two vertices in different sets are connected. However, in this paper we do not consider this notion in this context.



Connectivities Properties
Nodes: x, y
Links: e, f, g, h
Connectivites: x < ef|gh > y
none Add

Connectivities List
((x<ef|gh>y))

Start the text coupling formulation by declaring the node and link type identifiers



Connectivities Properties
Nodes: x, y, z
Links: e, f, g
Connectivites: y < f > z
none Add
y<f>z)

Connectivities List

Start the text coupling formulation by declaring the node and link type identifiers

(a) Interface for X_n, X_ℓ and C. (b) Specifying connectivity relationships.

Fig. 14. Interface for Coupling text

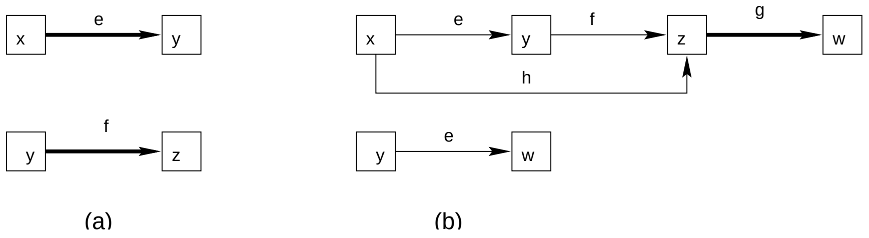
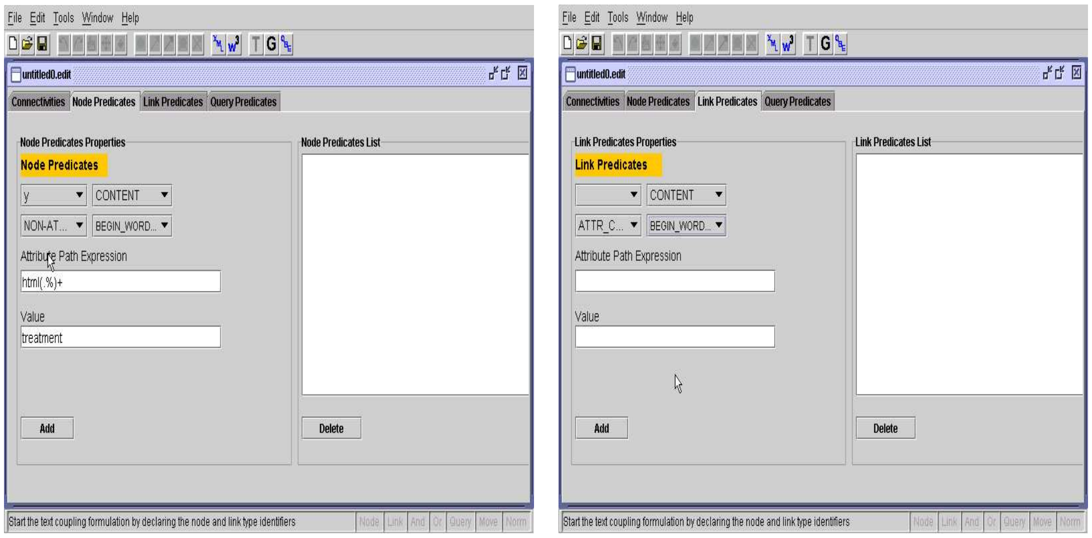


Fig. 15. Hybrid graphs.

connectivities C_3 as depicted in Case 1 in the preceding section. Next, we illustrate with examples how hybrid graphs can be used to resolve the shortcomings of drawing an OR or AND/OR-coupling graph.

Resolution of Case 1 AND-coupling graph can express connectivities which can be visualized as trees or graphs. Hence, the limitations discussed in Case 1 of AND/OR-coupling graph can be eliminated by drawing a hybrid graph containing AND-coupling graph. For instance, consider the connectivities C_3 and C_4 of Case 1 in Section 4.3.2. Queries containing these connectivities can be expressed by the hybrid graphs in Figure 13(a) and 13(b) respectively. Observe than these graphs are 2-connected graphs where all the components are AND-coupling graphs. Further, in each hybrid graph the AND-coupling graphs are in disjunction to one another. Similarly, the query containing the connectivities C_5 can be expressed by the hybrid graph in Figure 13(c).



Node Predicates Properties — Node Predicates: y, CONTENT, NON-AT..., BEGIN_WORD...; Attribute Path Expression: html(.%)+; Value: treatment. Node Predicates List.

Link Predicates Properties — Link Predicates: CONTENT, ATTR_C..., BEGIN_WORD...; Attribute Path Expression; Value. Link Predicates List.

Start the text coupling formulation by declaring the node and link type identifiers

(a) Screenshot for node predicates interface. (b) Screenshot for link predicates interface.

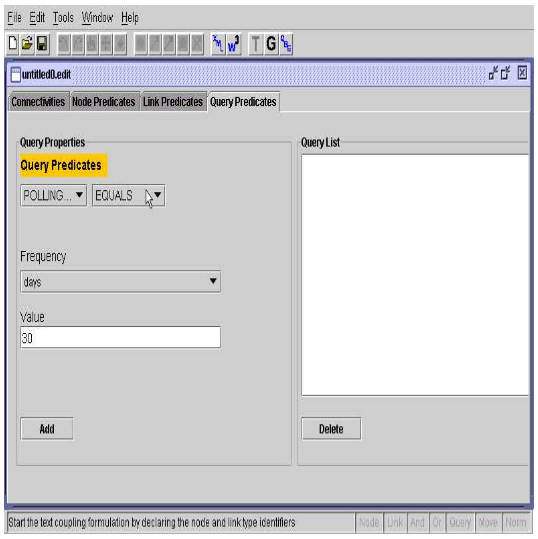
Fig. 16. Interface for Coupling text

Resolution of Case 2 We consider the case which is a limitation for both OR-coupling graph and AND/OR-coupling graph. Consider the query containing the connectivity C in OR-coupling graph as discussed earlier. This query can be formulated using a 2-connected hybrid graph as shown in Figure 15(a). Similarly, consider the query containing the connectivities C_6 as described in Case 2. This can be expressed by a 2-connected hybrid graph as depicted in Figure 15(b). Notice that it consist of an AND/OR and an AND-coupling graph.

Observe that in all the above cases the connected components are in disjunction to one another. Also, a pair of connected components always share some vertices with identical identifiers. This is because if G_i and G_j are two connected components such that $V_i \cap V_j = \emptyset$ then G_i and G_j can be expressed by two distinct coupling graphs. Hence, there is no need to express G_i and G_j using a hybrid graph.

5 Implementation of VISCOUS

We now discuss a visual query interface called VISCOUS that implements the two forms of query formulation techniques discussed in the preceding section.A preliminary version of this query formulation mechanism has been implemented in Java using JDK 1.2.2 and Swing.



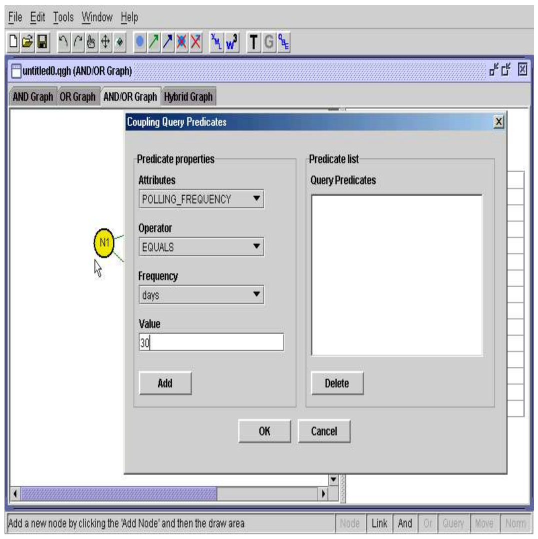
Query Predicates: POLLING... EQUALS

Frequency: days

Value: 30

Start the text coupling formulation by declaring the node and link type identifiers

(a) Interface in coupling text.



Attributes: POLLING_FREQUENCY

Operator: EQUALS

Frequency: days

Value: 30

Add a new node by clicking the 'Add Node' and then the draw area

(b) Interface in coupling graph.

Fig. 17. Coupling query predicates

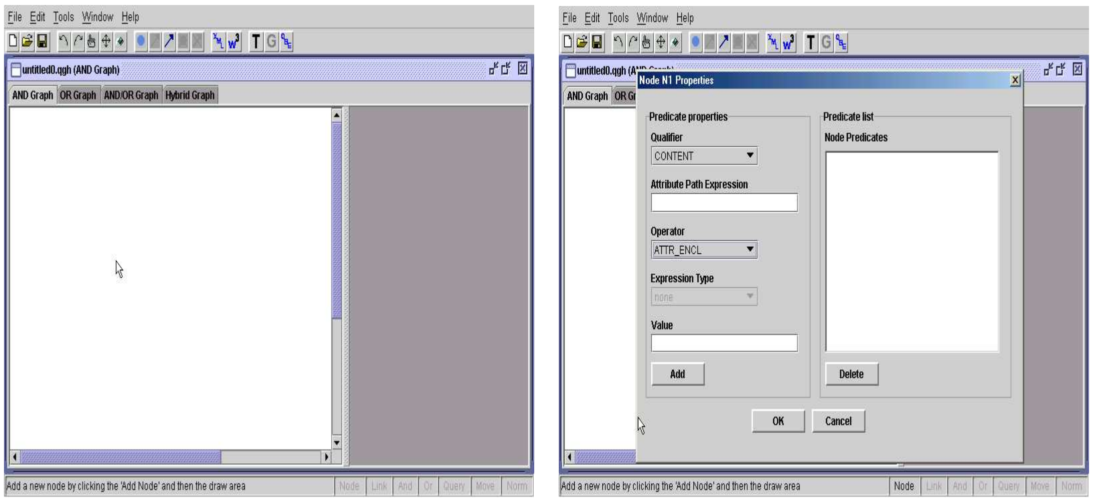
5.1 Formulating Query with Coupling Text

In coupling text, the user specifies the five components X_n, X_ℓ, C, P and Q in a GUI as shown in Figures 14, 16 and 17(a). Coupling text is a flexible query formulation mechanism and can be used to specify any meaningful query. We now briefly illustrate step-by-step how to formulate a query using coupling text. It must be noted that the walk-through serves as an outline for formulating a coupling query using the GUI of the coupling text. It is not intended to be an operation guide and therefore does not cover every function in the GUI.

(1) We first enter the node and link type identifiers in the GUI shown in Figure 14(a) by filling in the set of node and link type identifiers in the node and link type identifiers text fields respectively.

(2) Next, we add the connectivity set using these identifiers. Each connectivity is entered as shown in Figure 14(a). We select the source and target node type identifier using the node type identifier combo box. The link type identifiers are entered in the link path expression text field. The connectivity is then added to the collection of connectivities list. Figure 14(a) shows the screen after the above actions have been taken. The combo box labeled "None" is used to specify the relationships between the collection of connectivities (conjunction or disjunction) as depicted in Figure 14(b).

   Note that collection of connectivities in coupling text can be rather

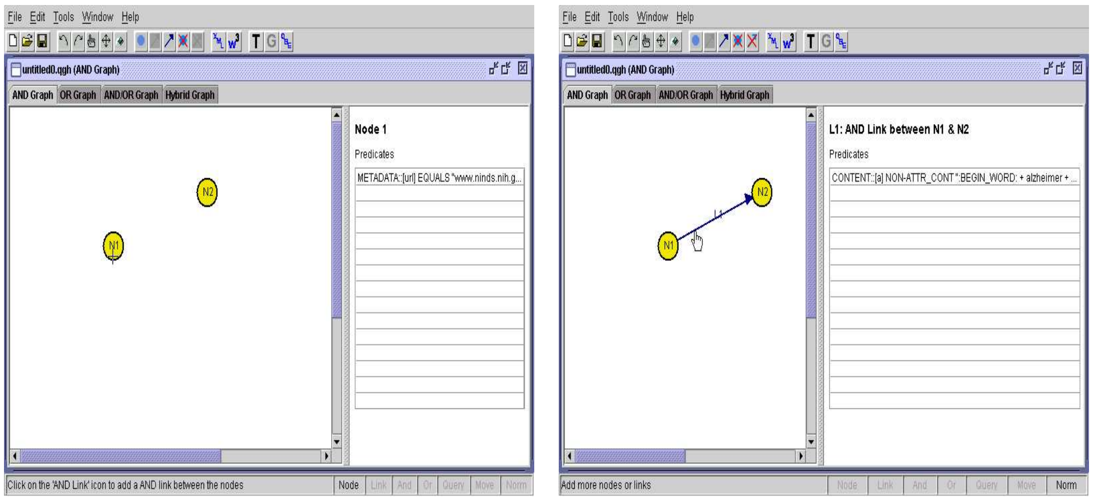


Qualifier: CONTENT
Attribute Path Expression:
Operator: ATTR_ENCL
Expression Type: none
Value:

Add a new node by clicking the 'Add Node' and then the draw area

(a) GUI for coupling graph. (b) Drawing a node in a coupling graph.

Fig. 18. Coupling graph



Node 1
Predicates
METADATA::[url] EQUALS "www.ninds.nih.g...

Click on the 'AND Link' icon to add a AND link between the nodes

L1: AND Link between N1 & N2
Predicates
CONTENT::[a] NON-ATTR_CONT ":BEGIN_WORD: + alzheimer + ...

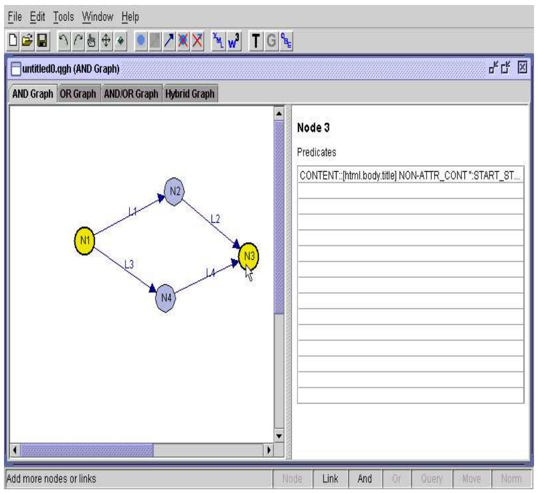
Add more nodes or links

(a) (b)

Fig. 19. Drawing a coupling graph.

complex; the number of braces and use of AND and OR relationships between the connectivity component can be intimating. For example, an attempt to edit a connectivity such as $(((a\langle e|f\rangle b \wedge c\langle d|f|g\rangle a) \vee (b\langle gh\rangle c \vee b\langle g\rangle a)) \wedge a\langle e\rangle z)$ is tedious and also error-prone. Therefore, we have implemented tools for the user to rearrange and amend the connectivity components.

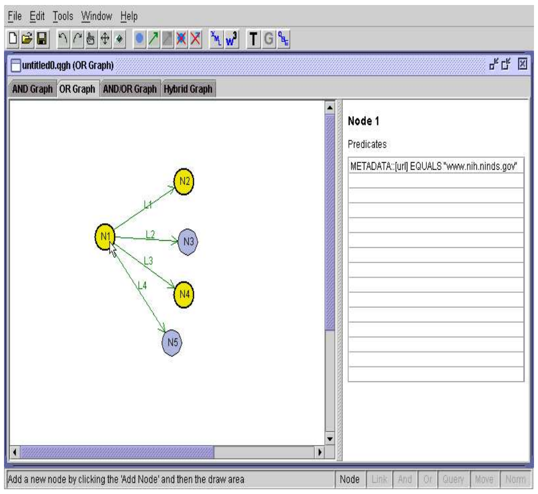
(3) The collection of connectivities that is required for a coupling query is completed at this phase. Next, the predicates on the node type identifiers



CONTENT::[html.body.title] NON-ATTR_CONT ":START_ST...

Add more nodes or links

(a) AND-coupling graph.



METADATA::[url] EQUALS "www.nih.ninds.gov"

Add a new node by clicking the 'Add Node' and then the draw area

(b) OR-coupling graph.

Fig. 20. Coupling graph

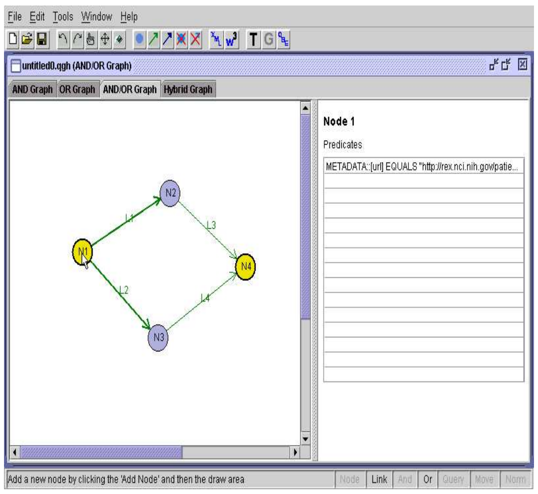
are entered. In order to do that, we need to switch from the "Connectivities" panel to the "Node predicates" panel. This is achieved by clicking on the"Node Predicates" pane on the coupling text tabbed pane. Figure 16(a) shows the GUI for entering predicates on node type identifiers.

(4) In order to add predicates on a node type identifier, the node type identifier is selected from the node type identifier combo box in Figure 16(a). Predicate qualifier is selected from the predicate qualifier combo box, and the predicate operator is selected from the predicate operator combo box. The attribute path expression and the value of the predicate are entered at the attribute path expression text field and value text field respectively. Figure 16(a) shows the screen obtained after completing the above-listed actions. Finally, the predicate is added to the predicate list by clicking the "Add" button.

(5) The predicates on the link type identifiers and the set of coupling query predicates are added similarly. Figure 16(b) and 17(a) shows the GUI for adding predicates on link type identifiers and coupling query predicates respectively.

5.2 Formulating Query with Coupling Graph

Next, we describe the second mechanism for formulating coupling queries, i.e., *coupling graph*. We show step-by-step how to formulate a coupling query using the GUI of AND-coupling graph.



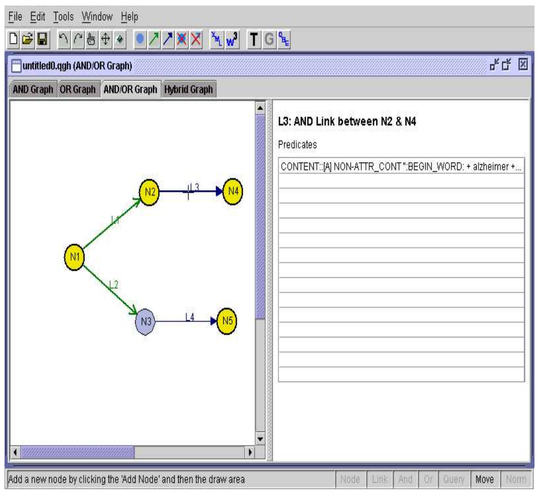
Node 1

Predicates

METADATA::[url] EQUALS "http://rex.nci.nih.gov/patie...

Add a new node by clicking the 'Add Node' and then the draw area

(a) AND/OR-coupling graph.



L3: AND Link between N2 & N4

Predicates

CONTENT::[A] NON-ATTR_CONT ":BEGIN_WORD: + alzheimer +...

Add a new node by clicking the 'Add Node' and then the draw area

(b) AND/OR-coupling graph.

Fig. 21. Coupling graph

(1) First step is to create a node in the drawing area of the coupling graph GUI (Figure 18(a)) by clicking the appropriate button.

(2) Next click at any position on the drawing area. A dialog box will appear as in Figure 18(b). Select the predicate qualifier and operator from the predicate qualifier combo box and predicate operator combo box respectively. Enter the attribute path expression and value into its respective text field. Finally, the "Add" button is clicked to add the predicate to the predicate list. Following which, the "OK" button can be clicked.

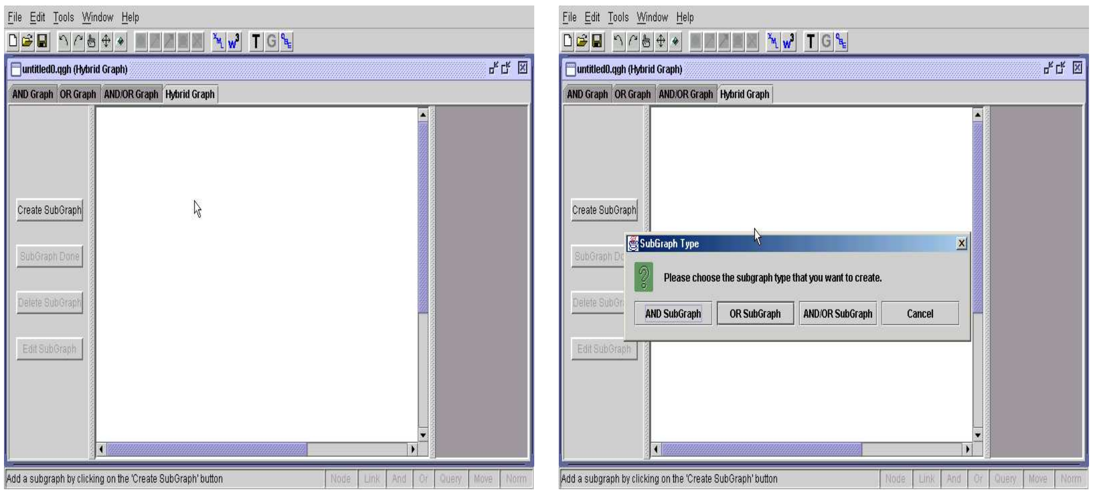
(3) The above step can be repeated to create another node. Figure 19(a) shows a screenshot of the creation of two nodes.

(4) Next, click on the "AND Link" button and then click on the node that is labeled N_1 and then the node that is labeled N_2. A link will be constructed between these two nodes as shown in Figure 19(b).

(5) To add or amend the predicates on the node and link type identifiers, double-click on the nodes and links whose predicates are to be modified. A predicate dialog box similar to the one shown in Figure 18(b) will appear for entering the components of the predicate on a node type identifier.

(6) The predicate for the query execution is entered using the coupling query predicate dialog box. This dialog box is accessed by selecting *Edit* and then *CouplingQueryPredicate* from the menu. The dialog box is shown in Figure 17(b). The procedure for entering the coupling query predicates is similar to the one that is followed in the coupling text interface.

The above procedure for drawing AND-coupling graph can be applied to draw a coupling graph as shown in Figure 20(a). Note that this graph expresses a



Please choose the subgraph type that you want to create.

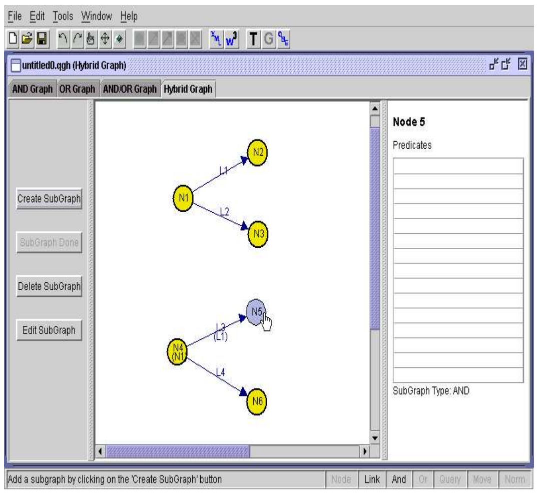
Add a subgraph by clicking on the 'Create SubGraph' button

(a) (b)

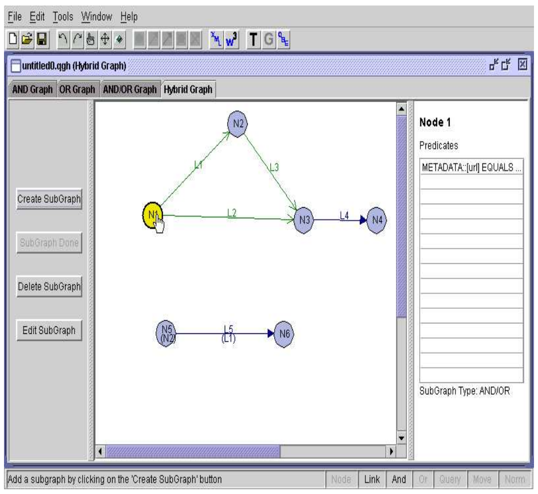
Fig. 22. Drawing a hybrid graph.

query with connectivities $(N_1\langle L_1\rangle N_2 \wedge N_2\langle L_2\rangle N_3 \wedge N_1\langle L_3\rangle N_4 \wedge N_4\langle L_4\rangle N_3)$ (identical to the AND-coupling graph in Figure 7(b)). We can follow similar procedure to draw an OR and AND/OR-coupling graphs. The difference is the selectivity of the "AND Link" and "OR Link" buttons in the different graph types. "OR Link" is available in OR-coupling graph drawing mode and both links are available in AND/OR-coupling graph drawing mode. Switching of graph type to be drawn is achieved by clicking on the respective graph type tabbed pane. For instance, Figure 20(b) is an example of OR-coupling graph expressing connectivities $(N_1\langle L_1\rangle N_2 \vee N_1\langle L_2\rangle N_3 \vee N_1\langle L_3\rangle N_4 \vee N_1\langle L_4\rangle N_5)$ (identical to the OR-coupling graph in Figure 8(a)). Figures 21(a) and 21(b) shows two AND/OR-coupling graphs expressing the connectivities $(N_1\langle L_1\rangle N_2 \wedge N_2\langle L_2\rangle N_4) \vee (N_1\langle L_3\rangle N_3 \wedge N_3\langle L_4\rangle N_4)$ and $(N_1\langle L_1\rangle N_2 \wedge N_2\langle L_3\rangle N_4) \vee (N_1\langle L_2\rangle N_3 \wedge N_3\langle L_4\rangle N_5)$ respectively (identical to the AND/OR-coupling graphs in Figures 9(b) and 9(a)). Note that in our GUI we represent an AND-edge and an OR-edge with blue and green arrows respectively. For example, in Figure 21(b) edges (N_2, L_3, N_4) and (N_3, L_4, N_5) are AND-edges. In Figure 21(a) the out-degree of vertex N_1 is two. Hence, edges (N_1, L_1, N_2) and (N_1, L_2, N_3) are OR-edges.

If the HYBRID graph type is selected, a window similar to that of Figure 22(a) is displayed. Clicking on the "Create Subgraph" button and a screen similar to Figure 22(b) will appear. Clicking on any one of these buttons will enable the user to draw the corresponding coupling graph. For instance, clicking on the "AND Subgraph" will allow us to draw an AND-coupling graph. The procedure for drawing a particular connected component in the hybrid graph is the same as discussed above. For example, Figure 23(a) represents a hybrid graph with two connected components consisting of AND-coupling graphs.



(a)



(b)

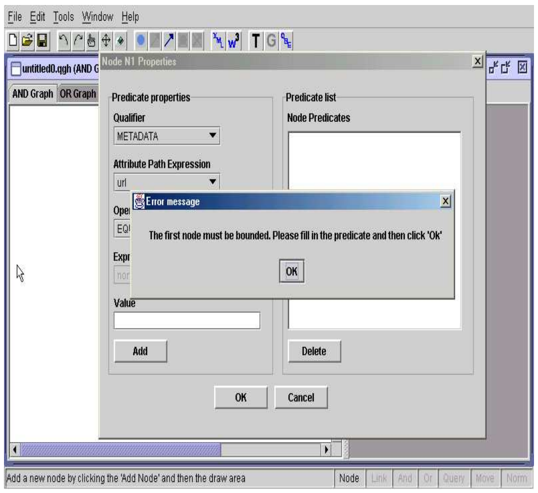
Fig. 23. Hybrid graph

This hybrid graph is identical to the one in Figure 13(c) where the node type identifiers x, w, k, y and z are N_1 (or N_4), N_2, N_3, N_5 and N_6 respectively and the link type identifiers e, h, f are L_1 (or L_3), L_2 and L_4 respectively. Similarly, Figure 23(b) is identical to the graph in Figure 15(b) where the node type identifiers x, y, z and w are N_1 (or N_5), N_2, N_3 and N_4 respectively and the link type identifiers e, f, g, h are L_1 (or L_5), L_3, L_4 and L_2 respectively. Note that it consist of an AND/OR and an AND-coupling graph.

5.3 Constraints

In Sections 2.4 to 4, we have elaborated on various constraints imposed on a coupling query. In this section, we discuss how some of these constraints are implemented.

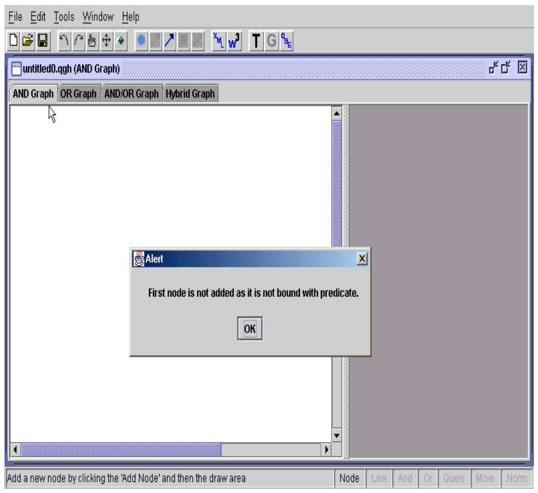
Bound source vertex As discussed in Section 2.4, the node type identifier for the source vertex in a coupling must be bound. That is, there must be at least one predicate applied on the node type identifier. We now show how this constraint is supported in a coupling graph in VISCOUS. Similar constraint is also supported in the coupling text mode. In coupling graph mode, the node predicate dialog box will appear automatically when the user attempt to add the first vertex on the drawing area (Figure 18(b)). Figure 24(a) shows the screen when the user click the “OK” button without adding any predicate to the predicate list. In this case, the user returns to the node predicate dialog box to enter the predicate. The screenshot in Figure 24(b) is fired when the



The first node must be bounded. Please fill in the predicate and then click 'Ok'

Add a new node by clicking the 'Add Node' and then the draw area

(a)



First node is not added as it is not bound with predicate.

Add a new node by clicking the 'Add Node' and then the draw area

(b)

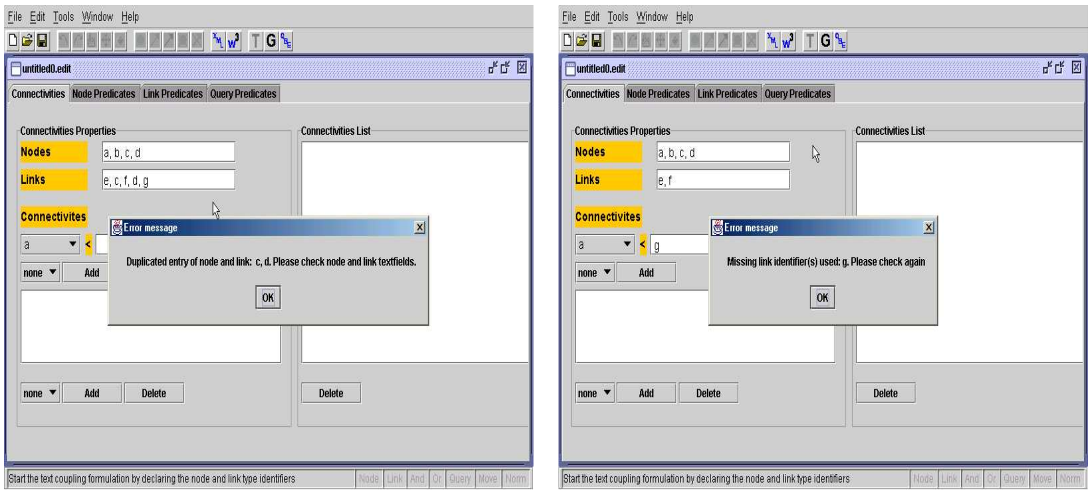
Fig. 24. Bound source vertex.

user click the "Cancel" button or close the node predicate dialog box. In this case, the user returns to the drawing area with no vertex created.

Constraints on X_n, X_ℓ and C As discussed in Section 2.4, the set of node and link type identifiers must be disjoint. Furthermore, the set of link type identifiers in connectivity set must be contained in X_ℓ. These constraints are satisfied when a coupling query is formulated in the graphical form, i.e., coupling graph. However, these constraints can be breached when the query is formulated in textual form, i.e., coupling text. In a coupling graph, the vertices and edges that are added to the drawing area by the user will be issued unique identifiers by the system. Furthermore, the problem of getting a mismatch between the set of node and link type identifiers used in the collection of connectivities with that present in the graph will not occur. This is because the node and link type identifiers found in the collection of connectivities are extracted from the graph consisting of vertices and edges added by the user.

In a coupling text, the user can decide the node and link type identifiers to be used. Therefore, there is a possibility for the user to inadvertently define an identifier x such that $x \in (X_n \cap X_\ell)$. To prevent such scenario, VISCOUS checks for common type identifiers in X_n and X_ℓ. If an identifier is used in both X_n and X_ℓ, then the user will get an alert message that is similar to the one shown in Figure 25(a).

Also, the user is only allowed to select the node type identifier for the connectivity from a combo box containing the list of node type identifiers entered into the set (Figure 14). As coupling text support complex connectivities, we



(a): Nodes: a, b, c, d — Links: e, c, f, d, g

Error message

Duplicated entry of node and link: c, d. Please check node and link textfields.

OK

Start the text coupling formulation by declaring the node and link type identifiers

(b): Nodes: a, b, c, d — Links: e, f — Connectivites: a < g

Error message

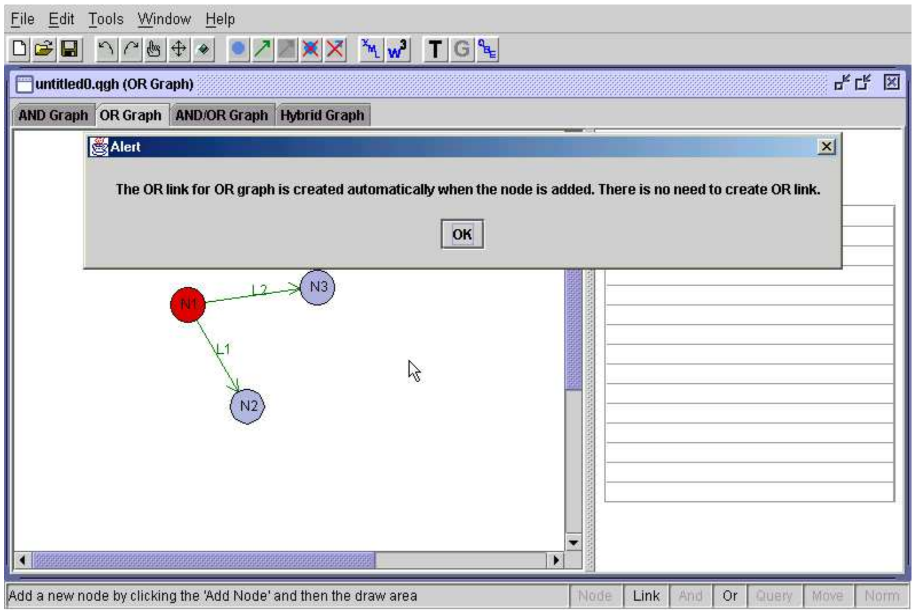
Missing link identifier(s) used: g. Please check again

OK

Start the text coupling formulation by declaring the node and link type identifiers

(a) (b)

Fig. 25. Constraints on node and link type identifiers.



Alert

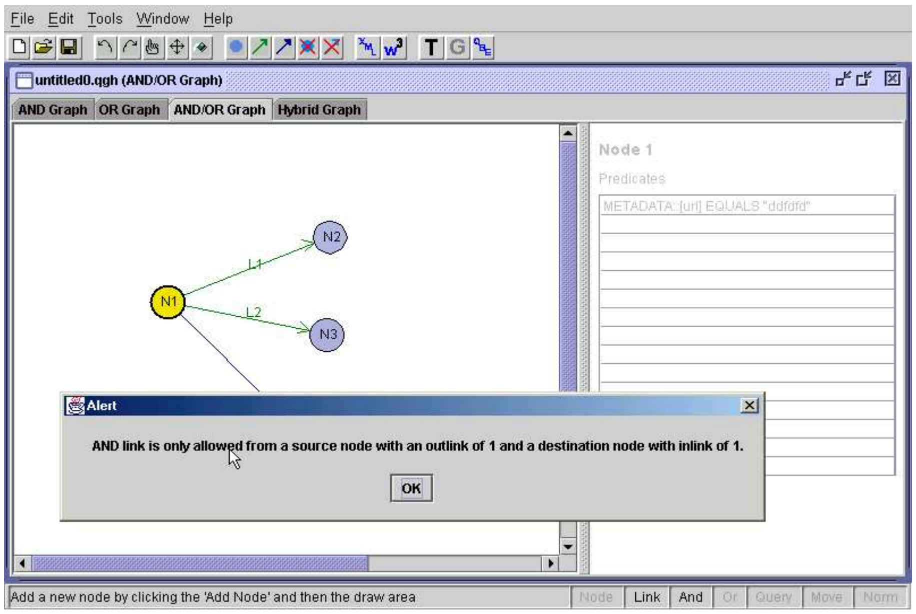
The OR link for OR graph is created automatically when the node is added. There is no need to create OR link.

OK

Add a new node by clicking the 'Add Node' and then the draw area

Fig. 26. Constraints on OR-coupling graph.

cannot apply the same technique to the link type identifiers. Consequently, the user may use an undefined node or link type identifier, say y, such that $y \notin X_\ell$. To prevent this, VISCOUS checks the link type identifiers entered in a connectivity. When an unknown link type identifier is used for a connectivity, the user will be informed through an error message that is shown in Figure 25(b).



Node 1

Predicates

METADATA::[url] EQUALS "ddfdfd"

AND link is only allowed from a source node with an outlink of 1 and a destination node with inlink of 1.

Add a new node by clicking the 'Add Node' and then the draw area

Fig. 27. Constraints on AND/OR-coupling graph.

Constraints on OR-coupling graph Recall that in an OR-coupling graph we only allow OR-edges. Furthermore, the depth of an OR-coupling graph is always one. These two features are supported in VISCOUS. In OR-coupling graph drawing mode we do not allow the user to draw edges. After the creation of the source vertex all the remaining vertices created by the user are automatically linked to the source vertex. Also, it is not possible to create an OR-edge from a node other than the source vertex (Figure 26).

Constraints on AND/OR-coupling graph As discussed in Section 4, in an AND/OR-coupling graph we disallow AND-edges from vertices whose in-degree or out-degree is more than one. Hence, we prevent the user to draw such invalid edges. For instance, Figure 27 shows an unsuccessful attempt to create an AND edge from vertex N_1. The edges L_1 and L_2 are OR-edges in Figure 27.

Constraints on hybrid graphs Informally, in the hybrid graph, there is a need to have at least one vertex in each connected component that is common in another connected component. In VISCOUS, this is achieved by the relabel dialog box as shown in Figure 28. Whenever a vertex that is the first in the second and subsequent connected components is drawn, the dialog box will appear to request the association of the new vertex with the others drawn earlier. Using this manner, the new vertex drawn will be logically the same as the one it is associated. However, it will be shown as two different vertices

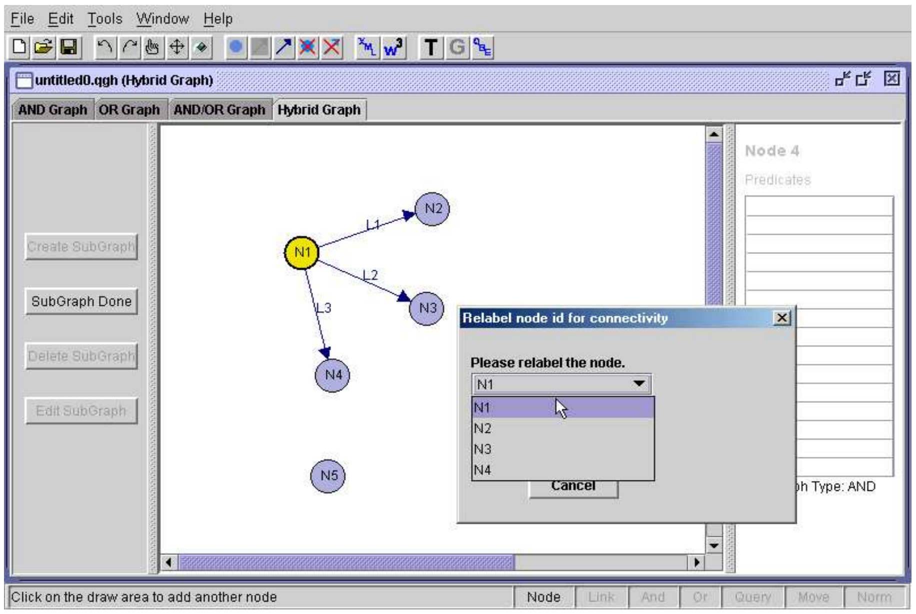


Fig. 28. Relabeling in a hybrid graph.

physically on the drawing area (Figure 23). Such a vertex can be identified by its dual labels. The label in braces refers to the node type identifier of the vertex that it is associated with. For example, Figure 23(a) shows a vertex that is associated with another vertex. Specifically, it can be seen that vertex N_4 is associated with vertex N_1. Similar the edges in each connected component can be relabeled. For instance, the edge labeled L_3 is associated with L_1 in Figure 23(a).

6 Conclusions & Future Work

In this paper, we described the formulation technique of a query mechanism in WHOWEDA for harnessing relevant documents from the Web. We express a web query in the form of a coupling query. We illustrated with examples how to formulate coupling queries in text form as well as pictorially using coupling text and coupling graph. Within this context we introduced two types of coupling queries, i.e., canonical and non-canonical queries. We also explored the limitations of coupling graph with respect to coupling text. We found out that AND, OR and AND/OR-coupling graphs are less expressive than their textual counterparts. To address this shortcoming we introduced another query formulation technique called hybrid graph which is a special type of p-connected coupling graph. Finally, we show how these formulation techniques are implemented in Java.

As part of our future work we wish to do the followings:

- Currently, coupling queries are directed connected acyclic graphs having single source vertex. We wish to generalize the coupling query into cyclic graphs with multiple source vertices. Also, we wish to augment coupling queries by allowing to impose conditions based on negation. Note that the inclusion of cycles and negation introduces interesting challenges with respect to the computability of the coupling query [2].
- Currently, we can either choose coupling text or coupling graph to formulate a query. We would like to make VISCOUS a two-way tool such that one can move from coupling text to coupling graph mode and vice versa while formulating a web query.
- We wish to develop a mechanism to estimate the evaluation cost of a coupling query over the Web. Such cost may help the user and query processor to optimize the cost of global web coupling operation.
- Finally, in this paper we have ignored how to formulate query for processing of forms in the Web. Many Web sites provide information by letting users fill out forms. Search engines do not fill forms autonomously as the number of possibilities is enormous, hence they are forced to miss interesting avenues that humans might follow. We wish extend our notion of coupling query to be able to autonomously fill out form and retrieve results by submission of the forms and manipulate these results further.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Weiner. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1):68-88, April 1997.

[2] S. Abiteboul, V. Vianu. Queries and Computation on the Web. *Proceedings of the 6th International Conference on Database Theory*, pp. 262-275, Greece, 1997.

[3] G. Arocena, A. Mendelzon. WebOQL: Restructuring Documents, Databases and Webs. *Proceedings of ICDE 98*, pp. 24-33, Orlando, Florida, February 1998.

[4] P. Atzeni, G. Mecca, P. Merialdo. Design and Maintenance of Data Intensive Web Sites. *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pp. 436-450, Valencia, Spain, 1998.

[5] S. S. Bhowmick. WHOM: A Data Model and Algebra for a Web Warehouse. *PhD Dissertation*, School of Computer Engineering, Nanyang Technological University, Singapore, 2001. Available at `www.ntu.edu.sg/home/assourav/` .

[6] S. S. Bhowmick, W. K. Ng, S. Madria. Imposing Disjunctive Constraints on Inter-Document Structure Using Coupling Queries. *Proceedings of the 12th International Conference on Database and Expert System Applications (DEXA'01)*, Munich, September, 2001.

[7] S. S. Bhowmick, W. K. Ng, S. Madria. On Formulation of Disjunctive Coupling Queries in Whoweda. *Proceedings of the 12th International Conference on Database and Expert System Applications (DEXA'01)*, Munich, September, 2001.

[8] S. S. Bhowmick, W.-K. Ng, S. K. Madria . Schemas for Web Data: A Reverse Engineering Approach. *Data and Knowledge Engineering Journal (DKE)*, 39(2), November, 2001.

[9] S. S. Bhowmick, W.-K. Ng, S. Madria. Representing Web Data in a Web Warehouse.*Computer Journal*, Oxford University Press, Under revision.

[10] S. S. Bhowmick, W.-K. Ng, S. Madria. Anatomy of a Coupling Query in a Web Warehouse. To appear in *International Journal of Software and Information Technology*, Elsevier Science, 2002.

[11] P. Buneman, S. Davidson, M. Fernandez, D. Suciu. Adding Structure to Unstructured Data. *Proceedings of the International Conference on Database Theory*, pp. 336-350, Delphi, Greeece, 1997.

[12] S. Cluet, S. Jacqmin, J. Simeon. The New YAT_L: Design and Specifications. *Technical Report*, INRIA, 1999.

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. A Query Language for XML. *Proceedings of the 8th World Wide Web Conference*, pp. 1155-1169, Toronto, Canada, May 1999.

[14] M. Fernandez, D. Florescu, A. Levy, D. Suciu. A Query Language and Processor for a Web-Site Management Systems *Proceedings in the Workshop of Semi-structured Data*, Tuscon, Arizona, May 1997.

[15] D. W. Embley. NFQL: The Natural Forms Query Language. *ACM TODS*, 14(2):168-211.

[16] T. Fiebig, J. Weiss, G. Moerkotte. RAW: A Relational Algebra for the Web. *Workshop on Management of Semistructured Data (PODS/SIGMOD'97)*, Tucson, Arizona, May 16, 1997.

[17] D. Florescu, A. Levy, A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Records*, 27(3):59-74, 1998.

[18] R. Goldman, J. McHugh, J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Proceedings of WebDB '99*, pp. 25-30, Philadelphia, Pennsylvania, June 1999.

[19] D. Konopnicki, O. Shmueli. Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QS System. *Theory of Database Systems (TODS)* , 23(4):369-410, 1998.

[20] L.V.S. Lakshmanan, F. Sadri., I.N. Subramanian. A Declarative Language for Querying and Restructuring the Web *Proceedings of the Sixth International Workshop on Research Issues in Data Engineering*, pp. 12-21, New Orleans, Louisiana, February, 1996.

[21] M. Liu, T. Guan, L. V. Saxton. Structured-Based Queries Over the World Wide Web. *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, pp. 107-120, Singapore, 1998.

[22] A. K. Luah, W.-K. Ng, E.-P. Lim. Locating Web Information Using Web Checkpoints. *Proceedings of the International Workshop on Internet Data Management (IDM'99), held in conjunction with the 10th International Conference on Database and Expert System Applications (DEXA'99)*, Florence, Italy, August 30-September 3, 1999.

[23] B. Ludascher, R. Himmeroder, G. Lausen et al. Managing Semistructured Data with FLORID: A Deductive Object-oriented Perspective. *Information Systems,* 23(8), 1998.

[24] G. A. Mihaila. WebSQL—A SQL-like Query Language for the World Wide Web. Master's Thesis, Department of Computer Science, University of Toronto, 1996.

[25] R. Weiss, B. Velez, M. Sheldon, C. Namprempre, P. Szilagyi, A. Duda, D. Gifford. HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering. *Proceedings of the Seventh ACM Conference on Hypertext,*, Washington DC, 16-20 March, pp. 180-193, ACM Press, New York, 1996.

[26] C. Zloof. Query-by-Example: A Database Language. *IBM System J.*, 16(4):342-343.